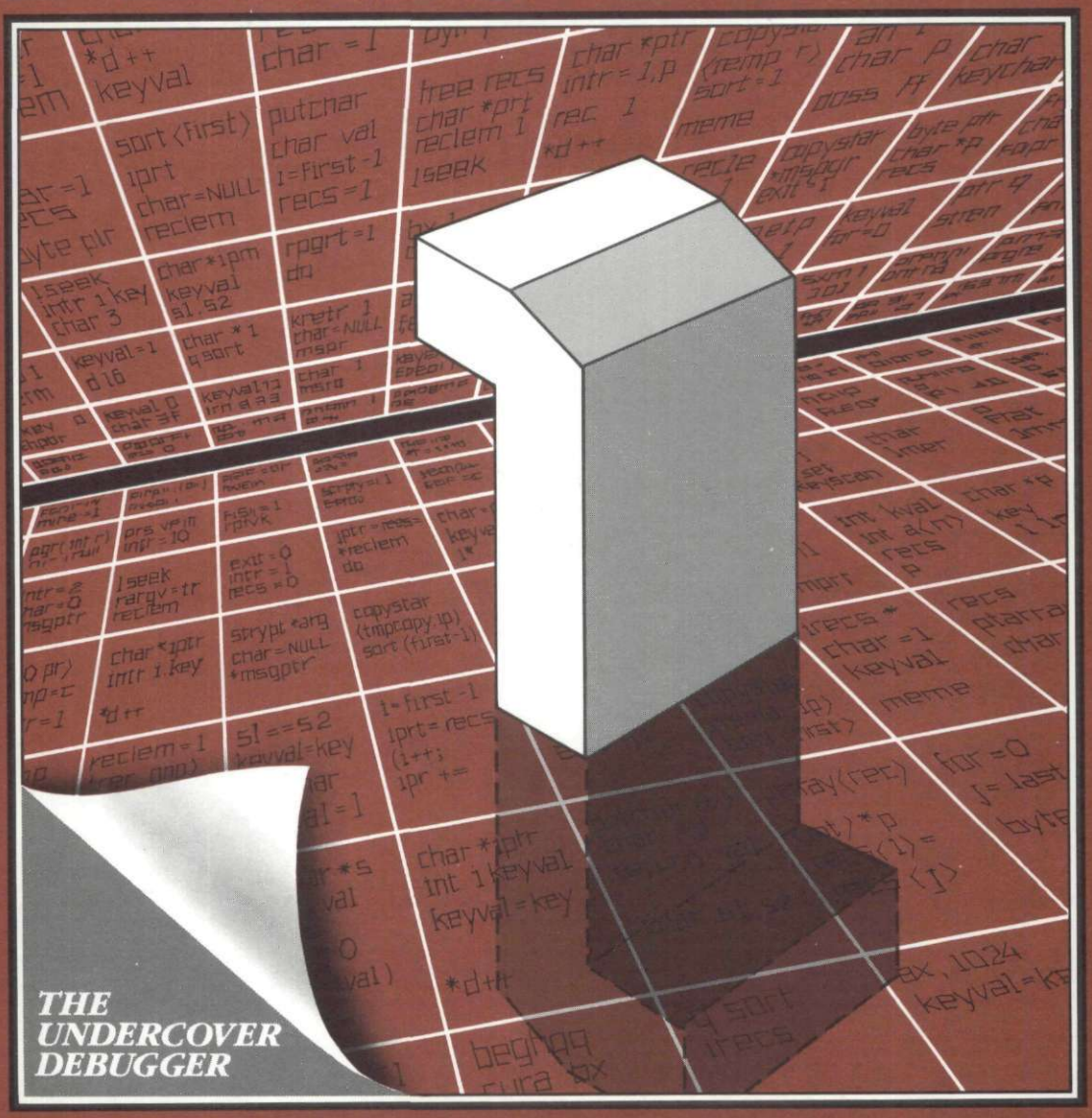
programmer's journal
The Resource Journal for IBM PC Programmers

# PERISCOPE




VOLUME 2
NUMBER 6

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

# POWER TOOLS

from *Phoenix*

You are a serious programmer. Your income depends on developing software solutions for complex problems. Fast. Error free. You need powerful tools for your compiler. That's what we sell.

## PLINK ™ 86
### OVERLAY LINKAGE EDITOR

- overlays for modules are automatically loaded during execution
- generates symbol tables for PFIX 86 PLUS
- accepts unlimited number of files
- capacity for over 35,000 symbol, commonblock and variable names
- change overlay structure without recompiling
- sorted memory map reports
- user defined width/height for reports
- define symbols as absolute addresses or offsets from other symbols
- over loader is user configurable

List: $395

## PFIX ™ 86 PLUS
### DYNAMIC SYMBOLIC DEBUGGER

- debugs PLINK 86 overlays
- view source code while debugging
- set temporary or permanent breakpoints
- advanced tracing and trapping capability
- in-line assembler for temporary patches
- preserves program screen and supports dual CRT configurations
- multiple windows to view code, data, breakpoints, register/stack contents, simultaneously
- supports 8087 numeric coprocessor

List: $395

## PMATE ™
### TEXT EDITOR PROCESSOR

- automatic disk buffering
- 10 auxiliary buffers that can be edited
- garbage stack
- user customization
- 120 comand macro language
- store macros permanently
- arithmetic and logical operations
- 100 numeric variables
- automatic formatting with indent and word wrap
- insert, overwrite and command modes
- horizontal/vertical scrolling

List: $225

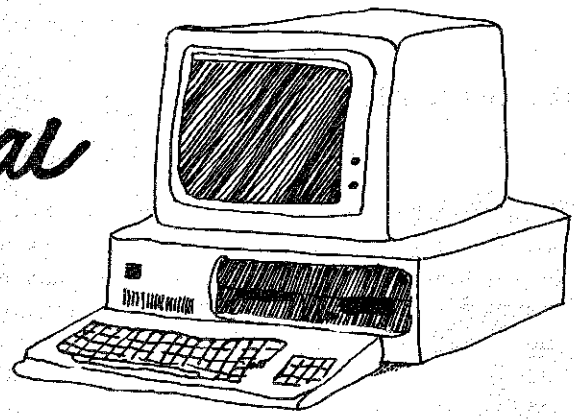
## $[PC]^2$ We Sell Power

Phoenix Computer Products Corporation
1416 Providence Hwy., Suite 220
Norwood, MA 02062
800-344-7200 (Mass.) 617-762-5030

Visa • Mastercard • American Express

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

# programmer's journal



## CONTENTS


HOUSEKEEPING . . . . . 2

LETTERS . . . . . 3

PJ INTERVIEW: PERISCOPE - BUILDING A HARDWARE BOARD FOR THE IBM PC . . . . . 5
*An interview with Brett Salter, producer of Periscope*

PJ FORUM: INTEL 80286 BUGS by FNE . . . . . 8
*Bug fixing in 1985*

REDIRECTIONS: AT&T SURPRISES ALL by Frank D. Greco . . . . . 12
*But is it IBM PC Compatible?*

C: FIELD USAGE AND POINTER SUBTRACTION IN C by Rex Jaeschke . . . . . 15
*Careful use of powerful features*

BASIC: CALENDAR FUNCTIONS FOR BASIC by Jehangir Gazdar & Roomy Shroff . . . . . 17
*Date calculation modules for your programs*

ASM: MEMORY SEGMENTATION by Bill Salkin . . . . . 23
*Know your way around the 8088 Segments*

PASCAL: PASCAL OBJECT CODE SIZE SOLUTION from Blaise Computing . . . . . 31
*Drastically reduce your EXE file sizes*

TOOLS: CORRECTING OVERSIGHTS IN DEBUG.COM by Daniel J. Jones . . . . . 33
*Enhancing Debug and other tools to increase your productivity*

dBASE II: dBase II to dBase III? by Ronald Moak . . . . . 39
*Is the upgrade really worth the trouble?*

REVIEW: THINKTANK by Tom Lichty . . . . . 43
*Would THINKTANK work well for you?*

MISC. . . . . 45

ADVERTISERS INDEX . . . . . 46


ISSN#0747-5861



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

## HOUSEKEEPING

My first love has always been programming. The job of magazine editor fills me with mixed emotions. It has its bad days, when I think about all the carefully polished code I otherwise might have written. I enjoy looking at lean code, watching what it does, and imagining how much an end user appreciates a well thought out program. When a whole week goes by and I haven't cracked my text editor or watched a series of compile and link statements sail past my screen, I honestly consider cashing in this editor's gig and resuming my first love, writing software.

But then something happens. A diskette mailer comes sliding under the door containing an unsolicited software article from a first time author/programmer. Well son-of-a-gun, many of us DO feel the same way about sharing hard-earned pieces of work. Everyone benefits from the experience. You have a piece of code you can immediately use or that illustrates a new technique, the first-time author feels good about putting in the extra time describing his routine to others, and I am not so depressed about the hours I might have spent writing software. It is an unspoken contract. You take the time to describe one of your routines, I'll temporarily set aside my programming tools to organize the journal, and those of you who have a "deadline this month" and won't be able to contribute, will send in money to pay the authors and the publishing costs.

It's not that I don't like editing a journal. It's just that *I LOVE PROGRAMMING A LOT*. You do too. And if you keep sending in your unsolicited software articles, then we'll all be in seventh heaven.

Peter Norton sent us a beta-test copy of his new utility series about to be released. They are just great. If you have a hard disk (XT or AT) they are absolutely essential. Snap up a copy. (We're glad to have you back, Peter. Writing books and articles have their rewards, but your expertise in both software and technical details has been sorely missed.)

For those of you who already have your PC AT's, would you kindly refrain from GLOATING so much on their new features? Give the rest of us a break. Remember, it's "be nice to a non-PC AT owner week."

— Greg

Articles should be forwarded to:

PJ/Editor
2765 Potter St.
Eugene, OR 97405
(503) 687-0398 □

*programmer's journal*

| | |
|---|---|
| Publisher: | Mary Carol Smith |
| Associate Publisher: | Elizabeth Oakley |
| Editor: | Greg Estes |
| Business Manager: | Steve Robinson |
| Copy Editor: | Joan Aschim |
| Production Manager: | Christine Baldwin |
| Production Assistant: | Jennifer Mullock |
| Cover Design: | Tom Kelly |
| Illustrations: | Julia King<br>Pamela Shute |
| Contributing Writers: | Ron Moak<br>Frank Greco<br>FNE<br>Jehangir Gazdar<br>Roomy Shroff<br>Rex Jaeschke<br>Bill Salkin<br>Daniel Jones<br>Tom Lichty |

| | |
|---|---|
| Published by: | Avant-Garde Publishing Corp. |
| Business Office: | Avant-Garde Publishing Corp.<br>1907 Garden Ave.<br>Eugene, OR 97403<br>(503) 345-3043 |
| Editorial Office: | 2765 Potter St.<br>Eugene, OR 97405<br>(503) 687-0398 |
| Subscription Inquiries: | Programmer's Journal<br>P.O. Box 1149<br>Wheaton, IL 60187<br>(503) 345-3043 |
| Postmaster send address changes to: | Programmer's Journal<br>P.O. Box 1149<br>Wheaton, IL 60187 |

Subscriptions: Bi-monthly rates are $24 U.S. ($30 in Canada, and $40 elsewhere)

IBM and Personal Computer are trademarks of International Business Machines, Armonk, New York; dBASE II is a trademark of Ashton-Tate.

Published bi-monthly.

Copyright © 1984
by Avant-Garde
Publishing Corp.



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ULTIMATE CLUB DISKETTES

The Houston Area League of PC Users (HAL-PC) has about 1,000 members in Houston. One of its big attractions is its library of public domain "shareware" (pay-if-you-like-it) software. A hard-working crew of volunteers from HAL-PC spends hundreds of hours a month acquiring, reviewing and cataloging new software, weeding out duplication, selecting the best programs to add to the library, copying the library and assisting users.

The public may get disks from the library for $2 each. For a library listing, send a self-addressed stamped envelope to HAL-PC software librarian, Nelson Ford, in care of "The Public Domain Library," P.O. Box 61565, Houston, TX 77208. Other user groups that would like to trade should write to the attention of Jack McClure, group coordinator, P.O. Box 610001, Houston, TX 77208. Software authors who would like to get widespread distribution of their public domain or shareware programs are encouraged to send them to Duane Hendricks, president, HAL-PC.

Nelson Ford
Houston, TX

YOUR OWN DISKETTE FORMATTING SERVICE, FMT.BAT

Stephen P. Goodman, via your local BBS, has put together the ultimate disk formatting batch file. In short, here is what you do.

1) Set up your RAMDISK
2) copy three files to it:
   FORMAT.COM
   FMT.BAT
   INPUT
3) Run FMT.BAT

The trick involves redirection of input and output. The FORMAT program pauses and demands some input. Why not just give it some input from a file? The second feature is sending the output from the FORMAT program to a NUL file; a piece of cake once you see the FMT.BAT file. In the dummy INPUT file you only have two lines which match your responses to the FORMAT requests. The first line is a carriage return, and the second is just the letter "n" which means you don't want to format another diskette.

The INPUT file looks like this

```
<CR>            .......carriage return
n               .......the letter n
```

The FMT.BAT file does the rest. Remember, this batch file starts rolling and formats diskettes in both the A and B drive. Just do a Ctrl-Break to stop the process.

The FMT.BAT file looks like this:

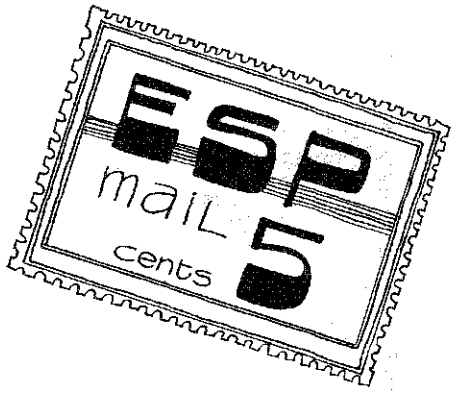
```
echo off
cls
echo Formatting Drive B:......
format b: <input >nul

echo off
cls
echo Formatting Drive A:......
format a: <input >nul

fmt
```

How about that for creatively using DOS features to solve a boring problem.

*From Stephen P. Goodman's BBS file and ESP Software*




COMMENTS ON DTACK'S VIEW OF UNIX

Regarding FNE's remarks in the DTACK GROUNDED article from Vol. 2 No. 4 of Programmer's Journal: I appreciate your comments on the flaws of the Unix Operating System; Unix has many to speak of. There are too many versions of the same operating system, file/record locking only exists on certain species of Unix, the sleep system call only allows a granularity of one second, the read system call on most versions cannot tell whether a character is available from the terminal or not, the file system is extremely vulnerable to transients, virtual memory schemes are all non-portable, the list can go on.



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

However, one should not criticize Unix incorrectly as you have done. You compare VersaDOS to Unix and say that "VersaDOS...if three programmers are using the same compiler, only one copy of the compiler is in the system (not true of Unix)..." This is hardly the truth about Unix. Most of the widely used commands, like ls, cat, ed, vi, and cc are setup as shared text programs, i.e., one copy of the program text is shared by many users. Any knowledgeable Unix system programmer knows that the Unix Link / Loader ("ld") has an option to produce shareable text.

You also say " . . . Everything in Unix is a file, the files talk to other files, and they are all on the disk!" Well, it is true that logically under Unix, everything is a file. Printers, terminals, disk drives, pipes, even memory is logically a file. Accessing memory is no different than accessing a file. That's one of the beauties of Unix; logical consistency in I/O. But they are not all on the disk as you are leading people to believe. Device files (terminals, printers, etc.) are really pseudo files in that by accessing the device file, actually activates a device driver in the nucleus of the OS. Therefore, the device file is merely a portal in which you indirectly access the kernel, which is in RAM, where the device drivers live.

In Unix, pipes are logically files like devices, and they exist only in RAM, not like the PC-DOS, MS-DOS 2.0 kludges. Where do you see a list of them on disk?

So go ahead and criticize Unix, it really does need it, but use some accurate information before doing some mud-slinging.

*Frank D. Greco*
*Kearny, NJ* ☐

# THE PROGRAMMER'S SHOP™

helps compare, evaluate, find products. Straight answers for serious programmers.

**SERVICES**
- Programmer's Referral List
- Compare Products
- Help find a Publisher
- Evaluation Literature free
- BULLETIN BOARD - 7 PM to 7 AM 617-826-4086
- Dealer's Inquire
- Newsletter
- Rush Order
- Over 300 products

**Free Literature - Compare Products**

Evaluate products Compare competitors. Learn about new alternatives. One *free call* brings information on just about any programming need. Ask for any *"Packet"* or "Addon Packet": ☐ ADA, Modula ☐ "AI" ☐ BASIC ☐ "C" ☐ COBOL ☐ Editors ☐ FORTH ☐ FORTRAN ☐ PASCAL ☐ UNIX/PC or ☐ Debuggers, Linkers, etc.

**"C" LANGUAGE**

| | LIST PRICE | OUR PRICE |
|---|---|---|
| MSDOS: C86-8087, reliable | $395 | call |
| Instant C - Inter., fast, full | NA | 500 |
| Lattice 2.1 - improved | 500 | call |
| Microsoft C 2.x | 500 | 349 |
| Williams - NEW, debugger | 500 | call |
| CPM80: Ecosoft C-new, solid, full | 250 | 225 |
| BDS C - solid value | 150 | 125 |
| MACINTOSH: Softworks | NA | 365 |
| Megamax-object, full | NA | 295 |

Compare, evaluate, consider other Cs

**BASIC** ENVIRONMENT

| | | | |
|---|---|---|---|
| Active Trace-debug | 86/80 | NA | 75 |
| BASCOM-86 - MicroSoft | 8086 | 395 | 279 |
| BASIC Dev't System | PCDOS | 79 | 72 |
| BASICA Compiler - BetterBASIC - 640K | PCDOS | NA | 185 |
| CB-86 - DRI | CPM86 | 600 | 439 |
| MACINTOSH COMPILER with BASIC syntax | MAC | NA | 325 |
| Prof. BASIC Compiler | PCDOS | NA | 89 |
| SCREEN SCULPTOR | PCDOS | NA | 119 |

Ask about ISAM, other addons for BASIC

**FEATURES**

GC-LISP - "COMMON" LISP, Help, tutorial, co-routines, arrays, thorough. PCDOS $475.

EXSYS - Expert System building tool. Full memory. Probability. Why. Intriguing. PCDOS $200.

**EDITORS** Programming

| | | | |
|---|---|---|---|
| BRIEF - Intuitive, flexible | PCDOS | NA | 195 |
| C Screen with source | 86/80 | NA | 75 |
| EDIX - Nice interface | PCDOS | 195 | 149 |
| FINAL WORD-for manuals | 86/80 | 300 | 215 |
| MINCE-like EMACS | PC/80 | 175 | 149 |
| PMATE-powerful | CPM | 195 | 175 |
| | 8086 | 225 | 195 |
| VEDIT-full, liked | 86/80 | 150 | 119 |

**UNIX PC**

| | | | |
|---|---|---|---|
| COHERENT - for "C" users | PClike | $500 | 475 |
| COHERENT-NCI-Realtime | PClike | 695 | call |
| XENIX - plus C to MSDOS | PC | 1350 | 1295 |

Ask about run-times, applications, DOS compatibility, other alternatives. UNIX is a trademark of Bell Labs

**LANGUAGE LIBRARIES**

| | | | |
|---|---|---|---|
| GRAPHICS: GraphiC-source | MSDOS | NA | 195 |
| GRAPHMATIC-3D. FTN, PAS | PCDOS | NA | 125 |
| HALO-fast, full-all lang. | PCDOS | 200 | 165 |
| FILE MGNT: BTrieve-all lang. | MSDOS | 245 | 215 |
| C to dBASE-source, caching | 8086 | 150 | 140 |
| Cindex + -source, no royal. | 86/80 | NA | 375 |
| CTree-source, no royal. | ALL | NA | 375 |
| dbC ISAM by Lattice | 8086 | NA | 250 |
| db VISTA-"Network", source | MSDOS | 495 | 465 |
| PHACT-up under UNIX, addons | MSDOS | NA | 250 |
| OTHER: Greenleaf-200 + | MSDOS | 175 | 165 |
| CSharp-**Real-Time** | MSDOS | NA | 600 |
| SOFT Horizons-Communicate | PCDOS | NA | 139 |
| SCREEN: CURSES by Lattice | PCDOS | NA | 125 |
| CView-input, validate | PCDOS | 245 | 195 |
| DATABURST-C, BASIC | MSDOS | 225 | 215 |
| MetaWINDOW-icons, clip | PCDOS | 150 | 139 |
| PANEL-many lang, term | MSDOS | 295 | 265 |

**RECENT DISCOVERIES**

**PERISCOPE** - PCDOS debugger - on own 16K, "Reset" box, symbolic, 2 screens. Debug after execute. $295.

**FORTRAN** ENVIRONMENT

| | | LIST PRICE | OUR PRICE |
|---|---|---|---|
| MS FORTRAN-86 - Impr. | MSDOS | $350 | $ 225 |
| DR Fortran-86 - full '77' | 8086 | 500 | 349 |
| PolyFORTRAN-XREF, Xtract | PCDOS | NA | 165 |

**OTHER PRODUCTS**

| | | | |
|---|---|---|---|
| Assembler & Tools-DRI | 8086 | 200 | 159 |
| Atron Debugger for Lattice | PCDOS | NA | 695 |
| CODESMITH-86-debug | PCDOS | 149 | 139 |
| Disk Mechanic-rebuild | MSDOS | 70 | 65 |
| HS/FORTH-fast | PCDOS | 250 | 239 |
| IQ LISP-full 1000K RAM | PCDOS | 175 | call |
| MacASM-full, fast, tools | MAC | NA | 100 |
| MBP Cobol-86-fast | 8086 | 750 | 695 |
| MicroPROLOG | PCDOS | NA | 285 |
| Microsoft MASM-86 | MSDOS | 100 | 85 |
| MSD Debugger | PCDOS | 125 | 119 |
| MultiLink-Multitasking | PCDOS | 295 | 265 |
| PC/FORTH + -well liked | MSDOS | 250 | 219 |
| PFIX-86 Debugger | MSDOS | 195 | 169 |
| PL/1-86 | 8086 | 750 | 495 |
| PLINK-86-overlays | 8086 | 395 | 345 |
| Polylibrarian-thorough | MSDOS | 99 | 95 |
| PolyMAKE | PCDOS | 99 | 95 |
| PROFILER-86-easier | MSDOS | NA | 125 |
| PROFILER-flexible | MSDOS | NA | 125 |
| Prolog-86-Learn, Experiment | MSDOS | NA | 125 |
| SCIL-control "Versions" | 86/80 | NA | 349 |
| SYMD debugger-symbols | PCDOS | NA | 119 |
| TLC LISP-86-full, liked | MSDOS | NA | 235 |
| TRACE86 debugger ASM | MSDOS | 125 | 115 |

Note: All prices subject to change without notice. Mention this ad. Some prices are specials.

Ask about COD and **POs**. All formats available.

**Call for a catalog, literature, and *solid value***

**800-421-8006**

**THE PROGRAMMER'S SHOP™**

128-J Rockland Street, Hanover, MA 02339.

Visa Mass: 800-442-8070 or 617-826-7531 MasterCard

1107




PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

## PERISCOPE — BUILDING A HARDWARE BOARD FOR THE IBM PC

*an interview with Brett Salter from Data Base Decisions, producer of Periscope, a symbolic debugger with a protected memory board for the IBM PC*

PJ: Brett, your two other products, Peeks n' Pokes and The Inside Track, are both sets of software routines that programmers can plug into their own programs and save hours of work. The readers of Programmer's Journal are fellow software developers and can easily see what it takes to produce and market a set of software routines: write the code, compose a manual, advertise the package and mail the thing out. Your newest product, however, involves a hardware board besides the associated debugging software. There are many other one-man software companies like yours that would tackle a software/manual product, but the thought of single-handedly manufacturing a piece of hardware must have presented some monumental problems. Could you describe how the project evolved, and what the trials and tribulations of designing a hardware board were like?

Brett: The project began when I was working on the code for The Inside Track, a set of lower-level routines, written in assembly language. This happened during the summer of 1983. In the process, I was hanging up the system a lot and using the little red button on the side of the machine a great deal. Debug was OK for what it offered, but trying to isolate problems in a piece of assembly code was consuming too much time due to the repeated rebooting of the machine. My first decision was to begin writing a better debugging tool. Even though I was happy with the way by new debugging software was progressing, I still found myself using the ON/OFF switch way too much. I knew there were sophisticated debugging/trace tools with hardware boards on the market, but their cost was prohibitive ($2500 I believe). I decided I wanted an inexpensive software debugging tool that also included a piece of hardware with some protected memory to save the state of the machine.

Building the board from scratch was a little less intimidating because I live in Atlanta, Georgia, which is the home of Hayes and Quadram, both of whom market hardware. I have some friends who had worked for these companies and after networking with a few of them, decided to tackle the project.

PJ: Did you have a vague idea of what you wanted or was it pretty well pinned down?

Brett: I was very specific about the kind of board that I needed. Many debugging boards (Big Blue's for example) are just one chip that issues an NMI (non-maskable interrupt) and jumps to the resident debugging software. No protected memory for saving the state of the machine or the ability to relocate the hardware address of the board and the software. Creating a reset switch is simple. You need only short two wires in the machine. Something else is definitely needed to provide more debugging tools. Periscope has 16K of protected memory, two reset switches (one remote and one on the back bracket), and two DIP switches to determine memory location. When the break-out switch is pressed, a non-maskable interrupt is generated and a flag is set to tell the machine that it is not a parity error, and the debugger is entered.

PJ: Whom did you contact once you had your specs for the board?

Brett: I talked with several people about doing the design, and was lucky to find an engineer at Georgia-Tech University who did consulting on the side. He did the work on an hourly basis, and it was quite reasonable. It took approximately two-and-a-half to three months to complete the first prototype, a wire-wrapped board.

PJ: Meaning a totally hard-wired board?

Brett: Yes. You start with a board that is filled with holes, stick the components through on one side and hard wire them on the other. It looks like a giant piece of spaghetti when you are done. As you isolate bugs you can just unsolder and rewire the connections. I had been working on the software since the summer of 1983 and in December of 1983 the design engineer began. About three months later (February of 1984) the first wire-wrapped prototype was done. It worked very well. I was adding software features from month to month and even though the first board took a little longer than I had hoped, things were going along quite well.

Once the first mock-up of the hardware board was done, the next step was to have some PC (printed circuit) boards made. These are bare boards without the components, but with the circuit patterns wave-soldered on them in layers. A local firm does this kind of small volume prototype board business. They did the artwork for the printed circuits, copying the first wire-wrapped version. Five PC boards were then produced, the chips were installed and they were heavily beta-tested. A few bugs were found and some minor enhancements were decided on, then five more prototype boards were made. These were again tested and a few problems were found. The circuit artwork was not quite right, and even though it seemed to be the prototype companies' fault, I split the cost with them for one more round of prototype boards. I wanted the final hardware boards for Periscope to be very high quality.

PJ: Were you responsible for the final testing of the hardware?



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

BRETT: To a certain degree. The beta testers and I ran the board through its paces. The prototypes were used in five different IBM PCs. The company that assembles the boards and installs the chips also tests and guarantees the final product to me. They have various equipment to verify the functioning of the components. All Periscope hardware is unconditionally warranted for a full year, and I even give a 30-day money-back guarantee on the product.

Once we completely tested the final five prototype boards to our satisfaction, it was time to order the first complete run of boards.

PJ: What is considered a small run?

BRETT: Well, I requested 250 to start with, which is not a particularly large order. The price you pay for the completed hardware is reflected in the volume you buy. If you buy 10,000 then you get a much better price break.

PJ: The whole process seems to have gone relatively smoothly. Didn't you have any major problems?

BRETT: I gained a tremendous respect for the purchasing agent of a company. The biggest frustrations, and learning experiences, occurred in this area. Development of the board design and software were going along so well that I got a little ahead of myself when it was time for the first production run of 250. I made endless phone calls searching for the chips and miscellaneous accessories (brackets, reset switches, mounting hardware).

I learned two very valuable lessons. First, you can line up about 90% of what you need but there always seems to be some minor part that just isn't available at the time you need it. If you are, say, looking for a metal bracket, you either have trouble tracking down the company that sells it, or their distributor is out of stock for many months. When I was ready to place the initial order for 250 boards, I had not absolutely nailed down all of the suppliers for all of the parts. So if you are planning to manufacture a piece of hardware, make sure you have made arrangements for everything down to the last little nut and bolt. You may find your whole production schedule delayed indefinitely while some company back orders a few left angle brackets for you. As the pressure increased from my difficulties with making the necessary part supply arrangements, my respect for the purchasing agents in large companies increased dramatically. Their job is a stressful one to say the least.

The other lesson I learned was that prices at the distributor level are up for grabs. You can really get down and start bargaining. If you call up a wholesale parts house and ask for a price on x number of computer chips, they will quote you a figure, but it won't be a particularly good deal. Now, if you have a good idea of what their discount price should be, you can call up and say that "I need x number of chips for x number of dollars, what can you do for me?" They are very willing to haggle at that level. Large quantity is a factor but smaller orders are still in the running. I remember calling up a distributor and ordering one 8088 chip to use with an 8087, and discussing the price. So big orders are not a prerequisite. After I talked with several distributors, I got a rash of calls at the end of their financial quarter. They had had rather flat sales for several months and wanted to really talk turkey with some prices to boost their quarter-end sales.

PJ: Would you be willing to develop a hardware board again?

BRETT: Sure. It was a rewarding experience even though the purchasing area had its headaches. I am not sure I would want to try it if I lived out in the boondocks. Having a number of friends here in Atlanta who were involved with hardware companies was a tremendous help. The important technical aspects of producing a board are being confident that the electronic design is solid and that the prototypes are thoroughly tested before starting a production run.

PJ: Any other high points in the Periscope hardware development process?

BRETT: We discovered one feature about the IBM PC hardware that we did not recognize before. My hardware engineer, who was laying out and testing the wire-wrapped version, experimented with different port numbers for the board. We understood that you could pick a port number as high as hex FFFF. Periscope needs four port numbers in a row. We picked some lower hex numbers and everything worked fine. Then the engineer tried using FFFF and he found that it conflicted with the disk drive. Apparently, IBM has set aside the six upper bits of the hardware port numbers for their use, so in reality you can only assign port numbers as high as 3FF.

PJ: When you developed Periscope's software, what are some of the major features you included?

BRETT: Besides the normal ones you find in Debug, an especially nice feature is the ability to do symbolic debugging. Once you are accustomed to telling your debugger to go to an address by name (G PRINTLN), the thought of requesting a location by number seems archaic. Periscope has on-line help, and your display screen is always preserved upon entering the debug mode. You can use both monitors if you wish. If you are debugging graphics routines you can watch your graphics on one monitor and use the debugging features on the other or vice versa. If you have ever tried to unscramble PSP's into the appropriate fields you will like the "record format" ability to display memory; Periscope unscrambles it for you.

PJ: Were you able to "boot-strap" your debugger software by using it to debug itself?



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

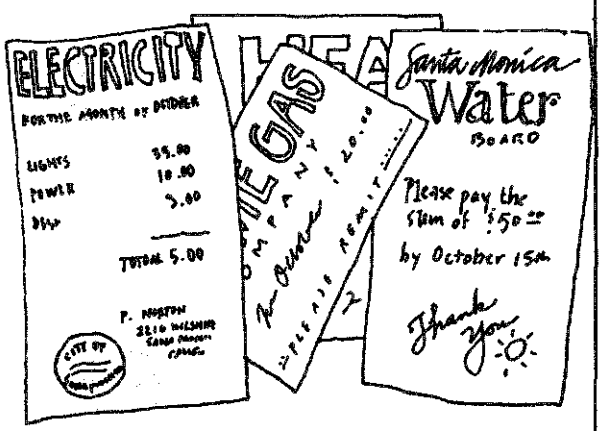
BRETT: After I wrote the core of the debugger, and got it to run fairly solidly, I would add on a new feature, assemble it and use the previous version to test it and find errors. It is an interesting process, using the tool you have already created to mold a better tool.

PJ: Sort of like robots making better robots. What kind of programming would be best suited for using Persicope?

BRETT: Any low-level programming in assembly as well as Pascal and C, or even BASIC if you interface with assembly routines. There is a Periscope user in Texas who is writing a complicated Chinese word processor (I'm serious), and he repeatedly needs to stop the program at a certain place and examine the values of his program variables. Since Periscope's software is resident, you can just run your program and jump out into the debugger whenever you like.

PJ: The information on designing and building a hardware board from scratch was much appreciated. For more details about Periscope (and bargaining at the OEM level) contact:

*Data Base Decisions*
*14 Bonnie Lane*
*Atlanta, GA 30328*
*(404) 256-3860* □




NORTON'S UTILITIES


Now with 8087* Floating Point Support

# *The* C Interpreter: *Instant-C*™

## Programming in C has never been Faster. Learning C will never be Easier.

***Instant-C***™ is an optimizing interpreter for the C language that can make programming in C three or more times faster than using old-fashioned compilers and loaders. The interpreter environment makes C as easy to use and learn as Basic. Yet ***Instant-C***™ is 20 to 50 times faster than interpreted Basic. This new interactive development environment gives you:

**Instant Editing.** The full-screen editor is built into ***Instant-C***™ for immediate use. You don't wait for a separate editor program to start up.

**Instant Error Correction.** You can check syntax in the editor. Each error message is displayed on the screen with the cursor set to the trouble spot, ready for your correction. Errors are reported clearly, by the editor, and only one at a time.

**Instant Execution.** ***Instant-C***™ uses no assembler or loader. You can execute your program as soon as you finish editing.

**Instant Testing.** You can immediately execute any C statement or function, set variables, or evaluate expressions. Your results are displayed automatically.

**Instant Symbolic Debugging.** Watch execution by single statement stepping. Debugging features are built-in; you don't need to recompile or reload using special options.

**Instant Loading.** Directly generates .EXE or .CMD files at your request to create stand-alone versions of your programs.

**Instant Floating Point.** Uses 8087* co-processor if present.

**Instant Compatibility.** Follows K & R standards. Comprehensive standard library provided, with source code.

**Instant Satisfaction.** Get more done, faster, with better results. ***Instant-C***™ is available now, and works under PC-DOS, MS-DOS*, and CP/M-86*.

Find out how ***Instant-C***™ is changing the way that programming is done. ***Instant-C***™ is $500. Call or write for more information.

**Rational** Systems, Inc.

(617) 653-6194
P.O. Box 480
Natick, Mass. 01760

Trademarks: MS-DOS (Microsoft Corp.), 8087 (Intell Corp.), CP/M-86 (Digital Research, Inc.), Instant-C (Rational Systems, Inc.)



# *Moving?*

Please send us both your old and new addresses, with Zip Codes. If possible, include the address label from a recent issue of *Programmer's Journal*. Allow three to four weeks for processing.

Programmer's Journal
P.O. Box 1149
Wheaton, IL 60187




PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

## INTEL 80286 BUGS

*by FNE, excerpted from DTACK GROUNDED*

A certain company is developing a product using the 286, and has paid $300 each for some supposedly good parts. But they discovered there are bugs in the interrupt handling hardware. Apparently the bug is related to the operation of the ready line during an interrupt acknowledge cycle. When they informed Intel, the response was "Oh, yes. We already knew about that!" We understand this bug will be fixed sometime during 1985(!).

Intel really stands behind its parts! Waaayyy behind...

[LATER: "The interrupt disable mask is ignored when the POPF instruction is executed with an odd stack pointer, unless two wait states are inserted in the fetch operation. The problem can be fixed by inserting two wait states into all data read operations [YE GODS! — FNE] or all even data byte read operations. Generally odd stack operations are not used by programmers because of the great speed penalty on an 8086/186/286. However, there is no additional speed penalty on the 8088 because it is always slow. Running a version of the less popular operating system revealed this bug.' — Understandably Anonymous]

U.A., are you asserting that there might be some 8086/186/286 software out there written for the IBM PC, which uses the 8088? We didn't know that! — FNE

IBM WATCH

You will recall that the original IBM PC, which came with an audio cassette interface for program and data storage, BASIC in ROM, and 16K RAM was really aimed at Apple Computer's home market and the Apple II, right? But that folks, seeing the IBM logo, immediately added enough memory and disk drives to turn the PC into a business machine which became vastly successful despite a deliberately busted keyboard to keep folks from using the PC as a word processor?

We did not have to tell you that Peanut (aka PCjr) was really aimed at the Apple Computer's home market and the Apple II because lots of other people (remember the WSJ's a rifle aimed at Apple's cash cow'?) did that for us. A few of those other people also asserted, as we did, that any machine with that IBM logo would be judged exclusively as a business machine and that the Peanut was doomed as a home (non-1-2-3) machine.

In the last issue we suggested that IBM was faced with the Apple III problem — lots of completed inventory (of PCjrs) gathering dust in warehouses. *Too much* inventory to be overlooked by the independent auditors which are legally mandated by SEC rules for publicly traded companies. (By the way, IBM's lawyers are much more experienced and professional than Apple's were three years back — our issue #4 caused a ruckus in Apple's henhouse back then).

IBM has just applied Apple's classic fix — bolting some new geegaws on the old turkey and in effect reintroducing it. This slides IBM by one audit and buys them a year before the next audit. The Apple III geegaw three years ago was the ProFile 5Mb hard disk and the Peanut geegaw is a very slightly improved keyboard and a lot of *very expensive* add-on memory modules. Enough add-on memory to take the Peanut out of the home computer price range (surprise?) and allow it to run 1-2-3 (*double* surprise?).

Let us emphasize this: IBM's first personal computer was intended to be a home computer, but because of its open architecture it was possible for third parties to expand it into a business machine whether IBM liked it or not. IBM's second personal computer was not only intended to be a home computer, but IBM (having learned from the PC) gave it a closed architecture and an abominable keyboard so that third parties *couldn't* turn it into a business machine. IBM has just backed down. Actually, they were *forced* to back down because of 350,000 Peanuts (our estimate) gathering dust in warehouses!

(There are now some IBM apologists running around asserting that Peanut was intended to be expandable all along. *Sure it was! That's why Peanut has to have an auxiliary power supply bolted on if you add more than just one memory module. You cannot even add a printer port without adding that auxiliary power supply (after adding one memory module).* Now you know whether to believe those IBM apologists.

As we told you once before, the revised Peanut is going to sell in very moderate quantities (while being hailed as a fantastic success by IBM's flacks) until the warehouses are emptied enough to quietly close out the line. You will recall that took three years in the Apple III's case.

Since *everybody*, including IBM, has written off the dinky diskless $595 "home" version of Peanut, we once again point out that IBM has now taken two swings at the home computer marketplace (and the Apple II) and has now struck out twice. (The first strikeout proved to be highly profitable for reasons which IBM had not anticipated.) Questions: is IBM dumb enough to try for the home market a third time with a personal computer bearing the IBM logo?



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Do not believe anything you read about the new Peanut being a $995 business machine. Add up all the accessories needed to run the most rudimentary business programs and see if you are not over $2000 without including the cost of a printer and the software itself. Remember, all of the cables cost extra (honest!). You can't steal the kids' TV for a monitor. Once you have added even *one* extra memory module you have to purchase that auxiliary power supply before adding *anything* else, including the printer interface (which costs extra). Check the price of those extra memory modules, but sit down first so you don't hurt yourself when you keel over. Check the tiny capacity of each of those memory modules. And *suure* you are going to get by on just one disk drive...

Once you have correctly identified the true price of the revised Peanut you will understand why IBM does not care whether sales of the PC are impacted.

POPGUN, er, POPCORN

IBM has rolled out its big gun, which turned out to be a popgun. Popcorn, now renamed the PC/AT, has just shot down exactly one personal computer now in the marketplace — IBM's own XT! The lowest price AT is more than double that of the PC clones (and the PC itself, for that matter). For $4000 you get *one* floppy and *no* video monitor or video board. And that is their cheapest AT, which we predict will sell (at that price) like coldcakes.

The $5800 version with a 20Mb hard disk and provision for three users is something else. Nobody believes you can do anything with that stripped $5800 version, but it will, when fleshed out into a $10,000 to $12,000 four-user system, give Fortune, Altos and some others fits. Mostly because of the IBM logo, but what else is new? A few rugged individualists and status-seekers will buy the fleshed-out $7000 version as a single-user package. (as usual, the prices we are talking about do not include a printer or software).

ALL HAIL THE CONQUEROR1

*As the might PC/AT dreadnought sweeps forth, devastating all who would challenge its powerful 80286 engine, it...* what's that? You ask whether the 6MHz 80286 used in the AT is faster than the 8MHz 80186 used in the Tandy 2000? Gee, we dunno. IBM claims that the AT is two to three times faster than its PC. Tandy has claimed for some time that its 2000 is two to three times faster than IBM's PC. Both of those claims look accurate to us. If you have been swallowing that blather about how powerful the 80286 is you should be ashamed of yourself. At 6MHz, it is about two-thirds of a 12.5MHz 68000. At 10MHz and 80286 would have an edge on a 12.5MHz 68000, but you have to remember that we have been shipping 12.5MHz 68000s for 27 months while it will be a year before you will see any 10MHz 80286s being shipped. (These figures are based on an average of a variety of different applications. As we have told you repeatedly, the 68000 and 80286 are each, for comparable clocks, superior to the other in specific areas. However — *6MHz is not a comparable clock!)*

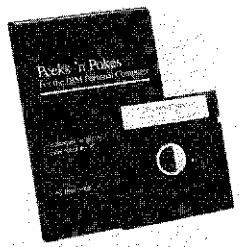
As far as we can see, the Tandy 2000 and the PC/AT have equal CPU capacity for single-tasking environments, which is the only sort of environment we are interested in. It is true that the AT's 80286 is well suited to multiuser/multitasking environments, but it is also true that the 286 runs a lot slower with that memory management unit switched on.

For $2995 you get a Tandy 2000 with 128K, two disk drives (1.44 Mb total) and a monochrome monitor and (the equivalent of) the "video monitor card". For $3995 IBM gives you a

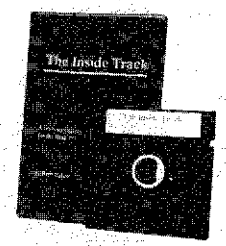

# Know Thy PC!

Are you writing programs in BASIC or Pascal? The popular **Peeks 'n Pokes** has a disk with 58 programs and a 38-page manual that helps you get 'underneath the covers' of the PC. Learn how to use PEEK, POKE, INP, OUT, and DOS/BIOS function calls to do what you want, fast! Do you want to perform functions not available from BASIC or Pascal? It's all explained in the manual and demonstrated in the sample programs. Source code included!

**Peeks 'n Pokes** shows you how to:
- Access the system's configuration
- Unprotect BASIC programs
- Scroll part or all of the screen
- Access the file directory
- Logically swap printers
- Read and change the keyboard
- Find more Peeks and Pokes
- And much more... for only $30.00




Want to know more? **The Inside Track!** is a collection of advanced utilities for the PC programmer. It contains a disk with 61 programs, a 42-page manual, and a fold-out memory map that helps you get better performance from the PC. With this package you can give your programs assembler-assisted speed from high-level languages, get control over memory, customize and control the PC, and more. Some programs require DOS 2.00. Source code included!




**The Inside Track!** shows you how to:
- Read/write files as fast as DOS
- Display data on the screen faster
- Reserve memory for your use
- Copy memory to another location
- Copy-protect your programs
- Load large programs faster
- Control the keyboard settings
- And much more... for only $45.00

MasterCard and VISA accepted. Shipping charges: $2.50 per order for UPS; $2.50 per item for First Class Mail to USA and Canada; $6.00 per item for Air Mail outside USA and Canada. Dealer inquiries invited.

Data Base Decisions • 14 Bonnie Lane • Atlanta, GA 30328 • 404/256-3860




PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

256K PC with *one* 1.2Mb floppy and no monitor or monitor card. With equal CPU power for single users, which do you think is the better deal? We both know which will sell better — the one with the IBM logo!

But the real message of the PC/AT is the chains and shackles for those three extra users. The AT will not be purchased because of what it can do for the user; it will be purchased to be *done to* the peons shackled to it. Almost all of the AT-specific software development for the 80286 is going to be aimed at the four-user version, and that software will wear four-user minicomputer price tags. Count on it.

You do remember that we have been telling you for 20 months now that the 80286 is best suited to locking several users into a separate memory spaces, don't you? You will notice that IBM figured that out, too...

BIG MUDDY

How many of you remember that less than a year ago there was just one IBM personal computer (the PC) for the clone-makers to shoot at? And only one operating system for the software vendors to target? IBM has (deliberately) done a good job of stirring up the mud. Already we read speculation about 80286 clones. And IBM has a briefcase portable and a file-server coming up — a file-server being a big disk networked to a bunch of PCs. Yes, we know Corona just introduced such a product, but it does not bear the IBM logo. That will stir up even more mud, and when the mud settles most of the clones will have disappeared.

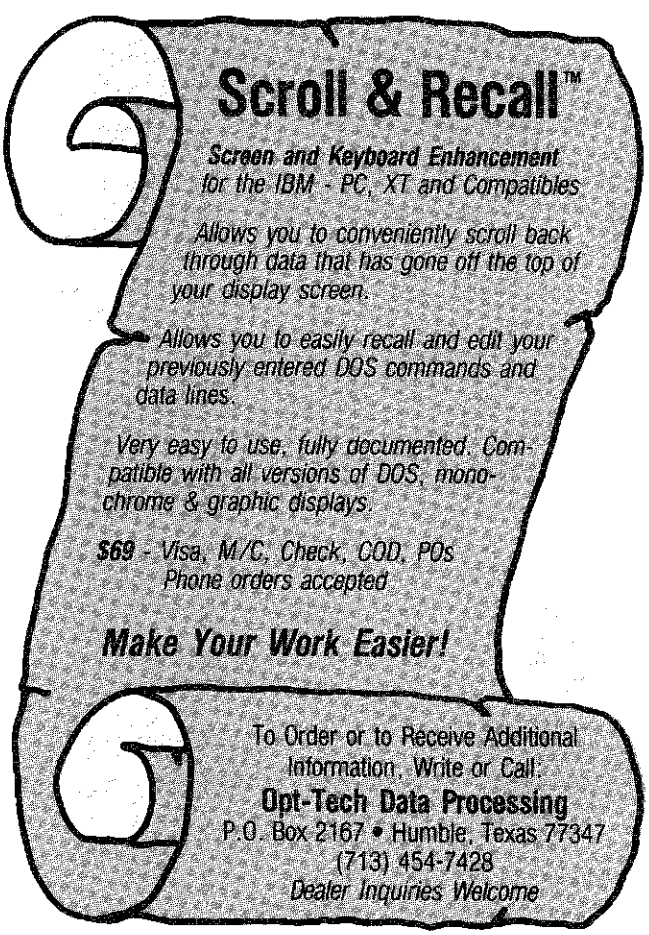
LETTERS

*"Correct me if I'm wrong, but I get the impression from reading DTACK that you don't like UNIX. Astounding if true, but I can only assume your dislike is from lack of technical information on it . . . You also seem to have some misconceptions about the world in general, and UNIX in particular. [7Umpty-eleventh exposition of 10%-90% rule deleted.8] . . let me direct you to an excellent book, The UNIX Programming Environment' by Rob Pike and Brian Kernighan. Unlike other books, this one shows you how you can do nearly everything in one command. (Also, Rob just bought his first house and can use the money.)*

*"My dad (same name, ex-JBL VP, now with Pioneer) says he has dealt with you before, and some of his acoustic friends (Ludwig Sepmeyer and Bruce Walker) say likewise. I called up Bruce Walker to find out if he was contributing to your mag, but he disclaims that and any schizoid condition that might have him doing so without his knowledge." Bart L., Murray Hill, NJ*

Bart, we do recall running into your dad at JPL but you have to realize that it was in reference to our somewhat largish speaker system, which features 12 each JBL LE-15A woofers and a pair of Altec 288C drivers and 311-90 horns. Like the current Motorola (68000) & Nat Semi (16081) brouhau, neither the JBL folk nor the Altec folk were especially enthused with our system! In fact, we remember your dad's 1967 transistor amplified design and have more recently seen his name on some digital audio technical committee reports (along with John Eargle, his former Altec counterpart). We have forgotten Ludwig and the original Bruce Walker — it has been 15 years since we built that largish speaker system, you know — and the current Bruce Walker ain't the same guy.

We have heavily condensed your letter re UNIX — and we warn future letter writers that the next pinhead who explains that damn 10%-80% rule to us again is going to be openly called a pinhead! You are not the first UNIX supporter to suggest that if we do not like UNIX, we must not know much about it. (We do appreciate you sending us yet another tech journal reprint.) In fact, we have studied the subject more than most of you think. The following reply is just an outline; the reader is expected to flesh out the argument on his or her own. — FNE



Scroll & Recall™
Screen and Keyboard Enhancement
for the IBM - PC, XT and Compatibles
Allows you to conveniently scroll back through data that has gone off the top of your display screen.
Allows you to easily recall and edit your previously entered DOS commands and data lines.
Very easy to use, fully documented. Compatible with all versions of DOS, monochrome & graphic displays.
$69 - Visa, M/C, Check, COD, POs
Phone orders accepted
Make Your Work Easier!
To Order or to Receive Additional Information, Write or Call:
Opt-Tech Data Processing
P.O. Box 2167 • Humble, Texas 77347
(713) 454-7428
Dealer Inquiries Welcome




PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

1) We unalterably support the concept of one man, at least one CPU.' Actually, we would like to discover a way to use multiple 68000s simultaneously for our sole and personal benefit. We regard folk — like UNIX types — who want to steal our CPU cycles with vast disfavor.

2) Philosophically, we are oriented toward the kinds of computers that one person will purchase for his or her own individual use. That is not UNIX.

3) When we consult with experts in smallish multi-user systems we are told that the PICK operating system is superior for business purposes while UNIX provides a superior environment for programming and documentation. Therefore it appears to us that UNIX is well suited to a well-defined but small portion of the multi-user market.

4) We insist that UNIX is so complex that it is best suited for full-time computer professionals. Since we are a hardware type and run a small company to boot, we are *not* a full-time computer professional. UNIX is therefore unsuitable for us.

5) UNIX systems require a *systems administrator!* This represents a step backwards to the bad old days of the mainframe priesthood in their white coats.

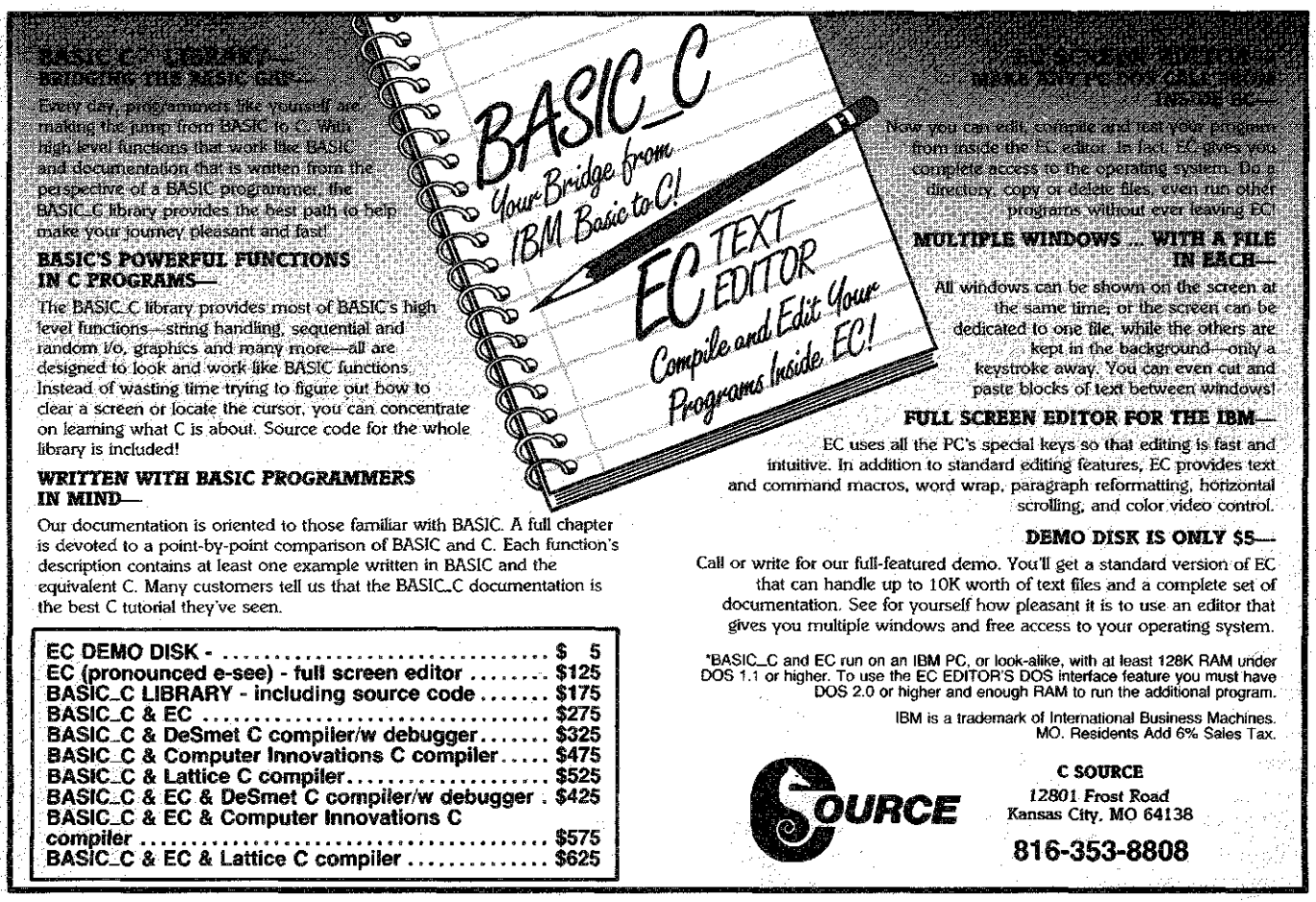
6) We are told that UNIX is wonderful because of the five megabytes of utilities that you don't have to re-write. Then we learn that virtually all UNIX systems on the market are emasculated, stripped of those wonderful utilities. The AT&T systems don't even come with a C compiler! What is so wonderful about emasculated UNIX?

7) *Which UNIX?* There are more varieties of UNIX than Carter has pills, and they are all mutually incompatible.

8) Multiple UNIX users can access the same file simultaneously. Wonderful! They can also all *modify* the same file simultaneously, which is less than wonderful.

9) We keep hearing about the legendary simplified UNIX which is going to arrive tomorrow, no lie. Question: if we get an operating system simpler than MS-DOS (available for $60) why do we have to pay AT&T a 77-lb. sack of gold for it? (A 22-lb. sack of gold for the emasculated version.)

10) Read (1) again. — FNE □


**BASIC_C: A BRIDGE — BRIDGING THE BASIC GAP—**

Every day, programmers like yourself are making the jump from BASIC to C. With high level functions that work like BASIC and documentation that is written from the perspective of a BASIC programmer, the BASIC_C library provides the best path to help make your journey pleasant and fast!

**BASIC'S POWERFUL FUNCTIONS IN C PROGRAMS—**

The BASIC_C library provides most of BASIC's high level functions—string handling, sequential and random I/o, graphics and many more—all are designed to look and work like BASIC functions. Instead of wasting time trying to figure out how to clear a screen or locate the cursor, you can concentrate on learning what C is about. Source code for the whole library is included!

**WRITTEN WITH BASIC PROGRAMMERS IN MIND—**

Our documentation is oriented to those familiar with BASIC. A full chapter is devoted to a point-by-point comparison of BASIC and C. Each function's description contains at least one example written in BASIC and the equivalent C. Many customers tell us that the BASIC_C documentation is the best C tutorial they've seen.

| | |
|---|---|
| EC DEMO DISK - | $ 5 |
| EC (pronounced e-see) - full screen editor | $125 |
| BASIC_C LIBRARY - including source code | $175 |
| BASIC_C & EC | $275 |
| BASIC_C & DeSmet C compiler/w debugger | $325 |
| BASIC_C & Computer Innovations C compiler | $475 |
| BASIC_C & Lattice C compiler | $525 |
| BASIC_C & EC & DeSmet C compiler/w debugger | $425 |
| BASIC_C & EC & Computer Innovations C compiler | $575 |
| BASIC_C & EC & Lattice C compiler | $625 |

**EC: A FULL SCREEN EDITOR — MAKE ANY PC DOS COMMAND FROM INSIDE EC—**

Now you can edit, compile and test your programs from inside the EC editor. In fact, EC gives you complete access to the operating system. Do a directory, copy or delete files, even run other programs without ever leaving EC!

**MULTIPLE WINDOWS ... WITH A FILE IN EACH—**

All windows can be shown on the screen at the same time; or the screen can be dedicated to one file, while the others are kept in the background—only a keystroke away. You can even cut and paste blocks of text between windows!

**FULL SCREEN EDITOR FOR THE IBM—**

EC uses all the PC's special keys so that editing is fast and intuitive. In addition to standard editing features, EC provides text and command macros, word wrap, paragraph reformatting, horizontal scrolling, and color video control.

**DEMO DISK IS ONLY $5—**

Call or write for our full-featured demo. You'll get a standard version of EC that can handle up to 10K worth of text files and a complete set of documentation. See for yourself how pleasant it is to use an editor that gives you multiple windows and free access to your operating system.

*BASIC_C and EC run on an IBM PC, or look-alike, with at least 128K RAM under DOS 1.1 or higher. To use the EC EDITOR'S DOS interface feature you must have DOS 2.0 or higher and enough RAM to run the additional program.

IBM is a trademark of International Business Machines.
MO. Residents Add 6% Sales Tax.

SOURCE

C SOURCE
12801 Frost Road
Kansas City, MO 64138

816-353-8808




PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

## REDIRECTIONS

*by Frank D. Greco, AGS Computers, Inc.*

AT&T SURPRISES ALL BY USING VENIX

VentureCom has recently announced that their Venix product is currently up to AT&T System V standard. Venix, a licensed AT&T Unix derivative, was previously based on the older System III plus some real-time and file/record locking goodies. VentureCom markets its product, now called "Venix / Encore", through the Massachusetts based Unisource Software Corporation (617-491- 1264). VentureCom has had plenty of requests for IBM AT versions, but another giant wants System V on their micros. AT&T bought between 10 to 20 copies of Venix for their PC 6300! It seems a little strange, but VentureCom licensed the source code from the phone company, then ported the kernel to the 8086 family, and is now selling Unix back to AT&T. In the process, VentureCom is recouping some (and eventually, all) of the $43K that they dished out to Ma Bell in the first place. Isn't this called "biting the hand that feeds you and enjoying it"?

PINK NOISE

In a desperate attempt to encourage people to buy video disks and video disk players, Pioneer Video Incorporated reduced the price of its model LD 660 helium gas laser disk player to $299. It originally was $599. You can guess that street prices for the Pioneer player will be around $200 with the cheap CED stylus based RCA model lurking nearby in the bargain bins. Pioneer also is offering 8 inch "shorter play" video disks at suggested retail prices of $10.99.

Panasonic has announced a device that allows you to cut your own optical disks. Has a .5 second access time (fast for video disks), an RS-232 interface, and 8 inch blank disks. All you supply is a video source, like a television, video camera, or computer video and Wham! Instant video disk!! Instant video disk company!! Finally, writable video disks are available (you can only write once). These disks are called WORM disks for Write Once Read Mostly disks. Don't blame me for that acronym; it's a generic term in the industry. Contact Panasonic's mother company, Matsushita Electric Corporation of America in Secaucus, New Jersey at 201-348-7000 for more info. I bet it costs a ton.

K&R

Thompson and Ritchie were featured in a recent issue of Fortune magazine (yes, I did say Fortune) and also in a recent issue of Microsystems. Apparently, the photos for both mags were taken the same day as these illustrious gentlemen had the same clothes on and the background was the same. If you look close at Dennis Ritchie's red tee shirt, you can read the inscription 'Bell Labs Sensitive Material' on the front. Thompson wore a similar shirt at a Unix seminar some years ago. From what someone told me, these computer scientists are mocking some internal rule that all sensitive materials were to covered with a bright red sheet that ostentatiously indicated proprietary information.

Formally speaking, Unix cannot be used as a noun. The AT&T legal beagles have said that Unix must be used as an adjective, as in the Unix Operating System. The reason for this is fairly obvious. AT&T does not want to lose the trademark for Unix. If Unix is used quite frequently in a generic sense for a pre-determined period of time, AT&T can lose its trademark. For example, this is precisely the reason that Xerox calls its machines "Xerox Copying Machines" and NOT "Xeroxes". Coca-Cola was also in danger of losing its "Coke" trademark in a similar manner. The words cellophane and cassette-player were once trademarks.

Did you ever wish you could get 16 shades of gray using your IBM graphics card? Avocado Computer sells an adapter that converts any IBM compatible graphics card to gray scale operation. It is called the Grayscaler and includes user manual with sample programs for plotting and, extensions to the 160 X 100 Low Resolution modes. Avocado's number is 6.02 * 10 E-23 moles per molecule . . . Whoops, that's AVOGADRO's number! Avocado's number is 714-528-1025.

Expect to see a low end microcomputer come from the Tandy folks real soon. The Tandy Texans will soon announce a Coleco Adam look-alike priced at $800. It will include 64K, one disk drive, and a printer. Its a jump ball on whether it will use a Z80 or a 8088 microprocessor.

PRINTER TIDINGS

A new technology for printers comes from the Liquid Crystal arena. Several copier companies have experimented with direct links from a computer to a super high resolution LCD flat panel inside of an ordinary copier. By shrinking the image and subsequently increasing the apparent resolution, the result is an inexpensive version of a laser printer! The little tricks that the Japanese engineers have pulled using clever optical effects to enhance the image should be applauded. I was stunned when I saw how clear the hard copy images were. Also, most japanese copier companies are manufacturing their machines so that they can be retrofitted with these LCD panels.



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

**Did you ever wonder how those young programming hot-shots do it?? Find out—in**

# EUREKA!

## ON THE IBM PC AND PCJR

## TIMOTHY KNIGHT

Learning to program your IBM PC isn't easy—especially with most of the dry how-to texts available today. But reading EUREKA! ON THE IBM PC AND PC*jr* is almost like having your very own tutor right at hand. Tim Knight, an 18-year-old with a dozen books to his credit, invites you to follow his thoughts and enter code with him while he first conceives, then designs, and finally implements a brand-new, state-of-the-art computer game. *Plus:* Step-by-step instructions for creating and using sound and graphics utilities for your IBM PC.

BRADY Computer BOOKS

FEBRUARY

Trade paper:
55940-0 • 224 pp. • $7.95

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

CAN YOU TOP THIS DEPT.

Hitachi announced a 4-megabit bubble memory device that has a 9.6 millisecond average access time. Most applications for bubble memory devices are for external memory, like replacements for hard disks in transportable or even lap computers of the near future. This Hitachi device is eight times faster than the PC XT winchester and four times faster than the IBM AT's hard disk. The fastest stepper motor winchesters are on the order of 40 millisecond average access time. Don't forget winchesters are electro-mechanical; bubbles are electronic (i.e., no moving parts and thus more reliable). Of course, the faster voice coil winchesters are available, but they are much more expensive than comparable bubbles. Samples of this amazing device are available now to hardware designers. Would be great for the tremendously disk-bound Unix operating system.

BLUE NEWS

When IBM unveiled the PC AT in Europe, they also reduced the price of their PC XT system unit 20%. This is usually an indication that IBM will lower the XT's price in the U.S. in a short period of time as all analysts are expecting.

IBM has told the press that the "AT" in their new PC AT stands for Advanced Technology. Believe it?.... I certainly don't. A close friend recently opened my eyes to some clever IBM marketing. It is so obvious, that probably most of you got the message subliminally (as IBM probably wanted). The letters "AT" is a direct attack at AT&T! You are now starting to mentally connect "AT" with IBM. When you say AT PC (or PC AT) your mind thinks of the IBM model; let's face it, the IBM PC world is part of our culture. There is enough subtle confusion between the AT PC (IBM) and the AT&T PC (Ma Bell) that IBM marketers are smirking more than a chesire cat in a tuna factory. I guess AT&T's "Watson, watch us now" slogan was not taken too lightly at IBM where Thomas Watson, Sr., started the company many years ago (and also instructed all his employees to "*THINK*") and his son, Thomas Watson, Jr., who successfully ran the company. T. Watson, recently passed away.

Maxine knew she was attractive, but still she anguished over THE REAL QUESTION: Was she PC-compatible?

Speaking of subliminal suggestion, there is a company out there that is selling a program that flashes a message of your choice up to 30,000 times a day on your PC or PC compatible's screen while you work on your daily tasks. Obviously, the message is printed so fast that your eye (or conscious mind) cannot "read" it, but your subconscious mind has "read" it. This is kind of self-hypnosis (or so they say) and supposedly works to some degree occasionally with selected people . . . hhhmmmmm . . . The company for those that are interested is Greetree Publishers (213)-483-5375 and the program costs $70. IBM has been using a similar approach to the subliminal method in its products. You know, those three little letters "IBM" printed in such a subtle and unobtrusive way before you start the IBM Personal Editor or IBM Wordproof programs. Sledgehammer instead of Subliminal suggestion . . . I guess . . .

At the recent Robots 8 show in Detroit, Michigan, IBM demonstrated a real-time kernel that was added to PC-DOS. This "kernel" (obviously not a kernel in the true sense of the word) permits users to merge the capabilities of IBM's PC-DOS with digital sampling capability. IBM showed three robots controlled by a PC XT and a Digital Input / Digital Output card. The real time kernel controlled the actions of the robots and also allowed PC-DOS to graphically display the status of all three robots. This so-called kernel samples data every 55.4 milliseconds. What IBM is calling a kernel is really just another layer between DOS and the Real World. Every 55.4 milliseconds, this "kernel" creates a pre-defined DOS interrupt which subsequently interprets the incoming digital information and performs the appropriate action.

I overheard the following conversation at a recent computer exposition:

"Did you hear about this new micro that was just announced?"

"No I haven't..."

"Well it has an Intel CMOS 80386 processor for the main cpu, 8 Megabytes of RAM, a 80387 floating point processor, bit-mapped screen with 2048 X 2048 resolution in 256 colors, three dimensional MACpaint look-alike, 8 channel sound in 16 octaves, portable, and has a flat pop-up 80 X 25 screen."

"Yeah . . ., but is it PC Compatible?" □



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

## FIELD USAGE AND POINTER SUBTRACTION IN C

*by Rex Jaeschke*

FIELDS IN C

Fields are special types of structure entries in that they are are less than an int (16 bits on the PC) and several of them can possibly be packed into an int without programmer intervention. See K&R pages 136 and 196 for specific details. Fields are useful for maintaining single or multiple bit flags in an int without having to use AND and OR masks to set and clear them. I have investigated the use of bit fields using three different compilers, namely DeSmet version 2.3 (DS), Microsoft version 2 (MS) and Whitesmiths version 2.1 (WS) and have found quite different results in their implementation. Test the following program with your compiler.

```
struct wordbit {
 unsigned f1 : 1;
 unsigned f2 : 2;
             : 3; /* 3 bit filler */
 unsigned f4 : 4;
 unsigned f5 : 5;
 unsigned f6 : 6;
             : 0; /* force int alignment */
 unsigned f8 : 8;
};

struct wordbit bf1;/* define static structure */

main ()
{
 bf1.f1 = 1;/* stores 1 in 1 bit */
 bf1.f2 = 0x3;/* stores 3 in 2 bits */
 bf1.f4 = 0xff;/* truncates & stores 15 in 4 bits */
 bf1.f5 = -4;/* truncates & stores 28 in 5 bits */
 bf1.f6 = 0377;/* truncates & stores 63 in 6 bits */

 /* f8 is not initialized explicitly */

 printf ("bf1.f1 is %u %d\n",bf1.f1,bf1.f1);
 printf ("bf1.f2 is %u %d\n",bf1.f2,bf1.f2);
 printf ("bf1.f4 is %u %d\n",bf1.f4,bf1.f4);
 printf ("bf1.f5 is %u %d\n",bf1.f5,bf1.f5);
 printf ("bf1.f6 is %u %d\n",bf1.f6,bf1.f6);
 printf ("bf1.f8 is %u %d\n",bf1.f8,bf1.f8);
}
```

First, none of the compilers likes the unnamed fields of length 3 and 0. K&R says these are valid for filler and int alignment respectively. This may cause you problems when porting to these compilers. Remove these two field definitions and recompile the program.

WS stores the bit fields as you would expect for the 8088, from bit 0 to higher order bits, or right-to-left. For some strange reason MS stores them left-to-right, although bits within each bit-field are stored right-to-left. DS checks field definitions and usage but assigns EACH bit field to its own int which defeats the whole purpose of using fields in the first place. (Note that the v2.3 release notes for DS show how to get bit fields packed densely using the non-standard format struct{unsigned bf1:3,bf2:2,bf3:4;}s.

All three compilers allow a bit field to be used as the operand to sizeof and return the length 2. K&R doesn't say if this is legal and I expect it's probably not. Anyway, it really doesn't make any sense to do so. MS won't allow a 16-bit field. Presumably, you should be using an int instead. DS lets you take the address of a field. Although this is illegal, it makes sense with DS as they allocate an int to each field so each field begins on an int boundary and does in fact have an address.

One thing not made clear in K&R is that fields can be used within the same structure along with "normal" structure variables and unions. For example,

```
struct nibble {
 unsigned nibble1 : 4;
 unsigned nibble2 : 4;
 unsigned nibble3 : 4;
 unsigned nibble4 : 4;
};

union bitfield {
 struct nibble n;
 int i;
} u;

main ()
{
 u.n.nibble1 = 1;
 u.n.nibble2 = 2;
 u.n.nibble3 = 3;
 u.n.nibble4 = 4;
 printf("u.i = %x\n",u.i);
}
```

This is a much simpler test to see how your compiler stores bit fields. While MS returns an answer of 1234, WS and DS (using the non-standard format) give 4321.

While bit fields may seem like a very good idea, and they are, there seems to be considerable differences in the way they are being implemented. So if you are accessing control bytes or words supplied by DOS or from some external source you should look at the generated code for your compiler to see how it allocates storage for fields so you can define your fields in the appropriate order.

POINTER SUBTRACTION

If we have two pointers, both of which point to objects of the same character array, and then take their difference, we have the number of objects separating them. That sounds reasonable and is in fact what K&R states. What they don't say is what happens if in p1 - p2 (where p1 and p2 are pointers), p1 points to an address less than p2. Will we get a negative difference or what? Let's try it.



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
/* problems with pointer subtraction */

main()
{
 char str[20],*c1,*c2,*c3;
 double darray[10],*d1,*d2,*d3;

 c1 = &str[2];
 c2 = &str[12];
 c3 = &str[52];
 d1 = &darray[0];
 d2 = &darray[4];
 d3 = &darray[14];
 printf("%5u - %5u = %6d objects\n",c2,c1,c2 - c1);
 printf("%5u - %5u = %6d objects\n",c1,c2,c1 - c2);
 printf("%5u - %5u = %6d objects\n\n",c3,c2,c3 - c2);
 printf("%5u - %5u = %6d objects\n",d2,d1,d2 - d1);
 printf("%5u - %5u = %6d objects\n",d1,d2,d1 - d2);
 printf("%5u - %5u = %6d objects\n",d3,d2,d3 - d2);
}
```

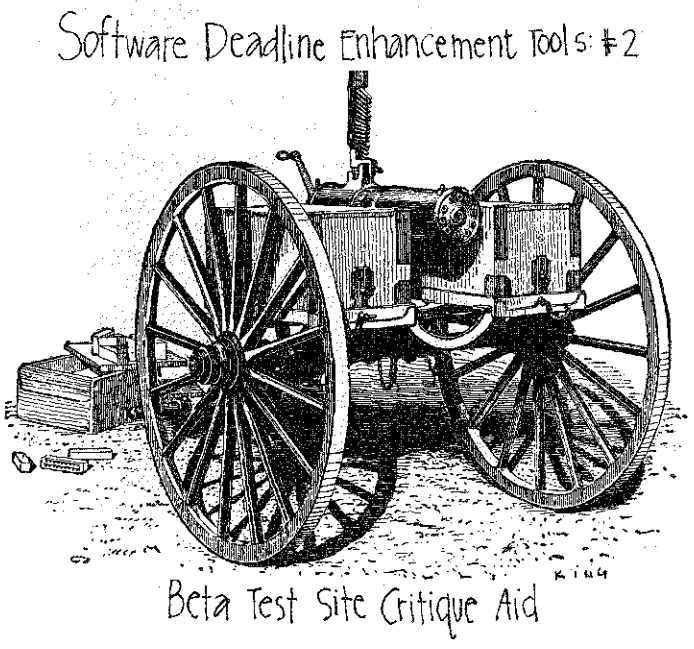
which produces

```
65500 - 65490 =     10 objects
65490 - 65500 =    -10 objects
    4 - 65500 =     40 objects

65440 - 65408 =      4 objects
65408 - 65440 =   8188 objects
65520 - 65440 =     10 objects
```

c1 and c2 both point to objects within array str so c2 - c1 makes sense and results in a pointer difference of 10 objects. c1 - c2 results in a difference of -10 objects. From this, one might assume that the pointers differ by 10 objects and that c1 points to an object nearer to the array start than does c2. In this case that just happens to be correct, but is it always true?

c3 points to str [52] which really doesn't exist, so c3 - c1 is meaningless. However, as there is no array bounds checking, we are told the pointers differ by 40 objects which would be correct if the array indeed was that large. str is an automatic variable array which is (with the DeSmet compiler and probably most others) created on the stack at runtime. Stack offsets are used to access such auto variables. As str was the first auto variable pushed on the stack and space for only 20 objects was reserved, a strange address of 4 was assigned to c3.




d2 - d1 indicates these pointers are separated by 4 objects. The numeric difference of addresses is 32 bytes but each double object is 8 bytes. C automatically scales the result to give objects rather than bytes by dividing the pointer difference by the pointer object size.

Whereas c1 - c2 gave us a reasonable result, d1 - d2 does not. We might have expected the result to be -4 indicating a difference of 4 objects with d1 having a lower address than d2 but this obviously is not the case. How can 65408 - 65440 equal 8188 objects? Remember that pointers are like unsigned integers so using unsigned arithmetic 65408 - 65440 equals 65504. Now 65504 / 8 (the object size) equals 8188 which is exactly what we got.

Note that we have evaluated the pointer expressions using unsigned arithmetic but we have displayed the difference using the signed decimal edit mask %d. Perhaps if we displayed it as %u, it would work. After all, we shouldn't display an unsigned result as a signed value. Let's see.

```
65500 - 65490 =     10 objects
65490 - 65500 = 65526 objects
    4 - 65500 =     40 objects

65440 - 65408 =      4 objects
65408 - 65440 =   8188 objects
65520 - 65440 =     10 objects
```

Nope, that didn't help any. In fact we have lost c1 - c2 as well. It seems that subtraction of one pointer from another in this case is meaningless and if you think about it, that makes sense although at first glance it is not obvious. However, all is not lost. As we can meaningfully compare two pointers we can still find the difference between them as follows.

```
/* pointer subtraction work-around */

main()
{
 char str[20],*c1,*c2,*c3;

 c1 = &str[2];
 c2 = &str[12];
 c3 = &str[5];
 printf("%d objects\n",c2 > c1 ? c2 - c1 : c1 - c2);
 printf("%d objects\n",c3 > c2 ? c3 - c2 : c2 - c3);
}
```

which generates

```
10 objects
7 objects
```

By determining which of the two pointers has the larger address value we can ensure that the result after their subtraction is always meaningful. □



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

## CALENDAR FUNCTIONS FOR BASIC

*by Jehangir Gazdar & Roomy Shroff*

When you first switch on your IBM Personal Computer, it slowly shakes itself awake, groggily tells you it thinks today is the first day of 1980, and asks if that is correct. If you have been generous to it (and yourself) and have given it a card with a battery operated clock/calendar, it wakes up knowing the correct date and time. But apart from stamping files with the date and time of their creation, it does nothing with this information.

It is hard to imagine a business application that does not require the entry and manipulation of calendar dates. Every transaction needs to be dated; financial programs need to know the number of days between dates; billing programs need to know if payments are overdue; inventory programs need to know when to re-order; all correspondence needs a date. It is therefore surprising that computer languages do not provide these facilities as intrinsic functions, especially as they are commonly to be found on cheap calculators and even on wristwatches. The BASIC subroutines described here were written to meet our need for these common calendar functions:

1. Day of the week of a given date.
2. Day of the year of a given date.
3. Days between two given dates.
4. Date a specified number of days from a given date.
5. Convert a numeric date into its spelled-out form.

Before a program can work with them, dates have to be typed in, and that is not as simple as it sounds. A computer is a literal beast, and the set format it demands for dates is uncomfortable for users. People do not use leading zeroes; the normal person will write "4/7/84", not "04/07/1984". There are also cultural booby-traps; what does that date mean to you? If you are American, you will say it is the seventh of April, 1984, but in many other parts of the world it would be the fourth of July.

Our computational routines expect the date in a rigid format; a ten-character string made up of a two-digit month, a delimiter, a two-digit day, a delimiter, and a four-digit year. They also assume that any date they are given is a valid calendar date, never having been taught "Thirty days hath September". Garbled formats and dates which never appeared on any calendar will be accepted and processed with the inevitable false results.

The program using them must therefore either use an input routine which rigidly enforces the required format, or accept a looser format and process it. It must also make sure that the input is an honest-to-goodness calendar date before passing it on. Whenever possible, we prefer to give the user some latitude in the way dates are typed, check the input for ambiguity, and convert it to the format required by the other subroutines. That is the task performed by the subroutine at Line 2000.

The delimiters commonly used in writing dates are the slash, "/", or the dash, "-". Our format-checking subroutine accepts as valid any string consisting of either one or two digits, either delimiter, another one or two digits followed by a delimiter, then either two or four digits for the year, and rejects anything else. Any dashes used as delimiters are converted to slashes; the PC accepts either delimiter when setting the system date, but other machines may not be quite so forgiving. Let us see how it works.

VALID is a variable used as a flag, and it is set to TRUE initially. A series of tests is carried out on DTE$ (the variable holding the date string), and if any test fails VALID is immediately set to NOT TRUE and the routine exited. The loop in Lines 2000 — 2010 checks that DTE$ contains only digits and delimiters; if any other character is present, VALID becomes NOT TRUE and the loop is terminated immediately. It is not a good idea to jump out of a loop leaving the NEXT pending, but needless looping can be avoided by setting the loop counter to the terminal value plus one. This is a safe method to terminate a loop prematurely.

Lines 2015 — 2020 change all dashes to slashes. Line 2025 searches DTE$ for slashes, and calls the position of the first one it finds DELIM1 and the second one DELIM2. Line 2030 looks funny if you are unused to BASIC's logical operators, but it is a good way of replacing a whole slew of IF . . . THEN statements. Translated into English, it says, "if within DTE$ delimiter 1 is not in position 2 or 3 and delimiter 2 is not in positions 4, 5, or 6, or if there is not at least one digit, but not more than two, between them, or if the year is zero or there is a third delimiter, then DTE$ is not valid, throw it out".

Lines 2035 — 2040 add leading zeroes to the day and month if their length is less than two. Line 2045 says that if the length of DTE$ is 8, then it must be a two-digit year, so assume it is this century and put '19' before it. A two-digit year which is '00' is ambiguous (1900 or 2000?), so that was discarded earlier. Finally Line 2050 decides that if DTE$ is not exactly 10 characters long by now, it was all a mistake. Yes, it is a lot of code just to make life a little easier for the user, but making things simpler for the user should be an important program function.

We have so far only been concerned with the format; we have yet to check if DTE$ is a plausible date. Line 2100 first calls the



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

subroutine at Line 2200 to break up our carefully formatted DTE$ into separate numerical values for month, day, and year, and decide whether it is a leap year. Since this article is for an American publication, the subroutines expect the "mm/dd/yyyy" format; month first, followed by the day and year. But they are easily modified to accept the alternative dd/mm/yyyy format by interchanging DAY and MONTH in lines 2200 and 2205 and also in Line 2830.

In the Gregorian calendar, a leap year is any year which is evenly divisible by 4, unless it is a centesimal year (ending in "00"). Centesimal years are leap years only if they are evenly divisible by 400; 1700, 1800, 1900 are not leap years, but 2000 is. The leap year question is handled neatly by the Boolean logic in line 2215:

```
(YEAR MOD 4 = 0 AND YEAR MOD 100 > 0) OR YEAR MOD 400 = 0.
```

YEAR MOD 4 = 0 will evaluate as true (-1) if the year is evenly divisible by four. YEAR MOD 100 0 will evaluate as true if the year is not a centesimal one. ANDing these results will give a true value only if both conditions are met. In this case, the rest of the expression has to be NOT TRUE. Even ORing it does not affect the result — LEAP.YEAR will be true and equal to -1. If the expression within the brackets evaluates as false (0) then LEAP.YEAR will be true only if the year is evenly divisible by four hundred.

Back in Line 2100, VALID is initialized to NOT TRUE, and then Line 2105 checks that MONTH is a value between 1 and 12, jumping out of the subroutine if it is not. Line 2110 checks for the special case of MONTH being February and DAY being 29. For this to be a valid date, the year must be a leap year. The special case of February 29 taken care of, Line 2115 first makes sure DAY is greater than zero, then extracts the maximum number of days permitted in MONTH and checks that DAY does not exceed that value. The '+' signs in the quoted string here and in those for the names of months and days are just fillers, easier to see than spaces if you are typing in the code from a listing. If DAY is within bounds, VALID is made TRUE and control returns to the program.

We need a little background on the calendar we use if are to understand the date routines. From the time of Julius Caesar, Europe used a calendar based on a solar year of 365.25 days. This Julian calendar added an extra day to every fourth year; "common" years were 365 days long, and "leap" years were given an extra day. But the Julian year was approximately eleven and a quarter minutes too long, which works out to an error of about 3 days in 400 years. To correct this error, Pope Gregory XIII introduced a new calendar in 1582. This Gregorian calendar made 3 out of 4 centesimal years common years instead of the leap years they had been in the Julian calendar. Pope Gregory also corrected the accumulated error by declaring that the day following October 4, 1582 would be October 15, 1952 (and changed New Year's Day to January 1 from March 25).

This Gregorian calendar is the one in common use throughout the world today, though it is by no means the only one, nor was it adopted immediately by all countries. The British, for example, waited until 1752 before introducing it to its colonies, including America; Russia did not introduce it until 1918. Our calendar subroutines are for this Gregorian calendar. If you use them for dates in the distant past, be prepared to do a little research to see if this calendar was in use at the time and place in question, and apply a correction factor if it was not.

All the calendar functions are dependent on the day number subroutine starting at Line 2700. The algorithm used in this subroutine is a BASIC adaptation of the formula frequently published for date computations on Hewlett-Packard's programmable calculators. HP has always included a calendar functions program in the 'Standard Applications Pac' supplied with their magnetic-card programmables, though the validity of those programs is restricted to the usual 200-year range from March 1, 1900 to February 28, 2100. This restriction is common in many calendar routines; because 2000 is a leap year, the code to check centesimal years can be omitted, making the program considerably shorter. The full formula is published in HP's "67/97 Users Library Solutions" book on calendars. A similar formula appears in John Ball's "Algorithms for RPN Calculators".

This subroutine converts a date to the number of days (DAY.NUMBER) elapsed since a given date in ancient times (this is not the Julian Day number used by astronomers — more on that later). Once a day number has been calculated for a date, simple arithmetic provides the day number of a date any number of days before or since. Conversely, the difference in the day numbers of two dates is the number of days between them. The day of the week is determined by taking the day number mod 7. The subroutine at Line 2800 applies the same formula in reverse to convert a day number back to a date in our standard format.

To find the day of the week, the subroutine at Line 2300 first gets the day number of DTE$ by calling the routine at line 2700. Line 2305 obtains a numerical code for the day of the week, DOW, by taking DAY.NUMBER! MOD 7. We cannot use the MOD operator for this because that works only for integers, and DAY.NUMBER! is a single-precision value. The technique used here can be substituted for MOD whenever you use that operator if you are translating these routines to a BASIC which does not support it. DOW will be a number between 0 and 6, where 0 is Sunday, 1 is Monday and so on. Line 2310 uses this to extract the name of the day from the quoted string, and 2315 strips it of any trailing plus signs.

The subroutine at Line 2400 returns the number of days between two dates by the straightforward process of getting their day numbers and subtracting the second from the first. If you need to determine whether date two is earlier or later



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

than date one, remove the ABS function from Line 2410. A negative value will indicate that date two has a larger day number, and so must be later than date one.

To calculate the date any number of days before or after a given date, we use the subroutine at line 2500. It gets the day number of the given date, adds to it the number of days difference (negative value if you need a previous date), and then returns in DTE$ the date of the new day number by calling the subroutine at Line 2800 to convert it.

"May 7, 1984" looks better and is easier to read than "05/07/1984" and has the added virtue of being unambiguous. Your program can spruce up its output by using the subroutine at Line 2600 to do this conversion. Lines 2605-2610 extract the name of the month in the same fashion that the day name was extracted earlier, and Line 2615 puts it all together in ALPHA-DATE$. DAY.DATE$ adds the day of the week to make it "Monday, May 7, 1984".

The last subroutine, at line 2900, takes DTE$ and returns the day of the year in DOY. Line 2905 does the work quite simply. DTE$ is split up into DAY, MONTH and YEAR, and one is subtracted from the current month to get the number of previous months. This figure is multiplied by 31 and added to the days of the current month. From the result is subtracted a figure to compensate for the false assumption that every prior month had 31 days. If the current month is later than February and it is a leap year, one day is then added.

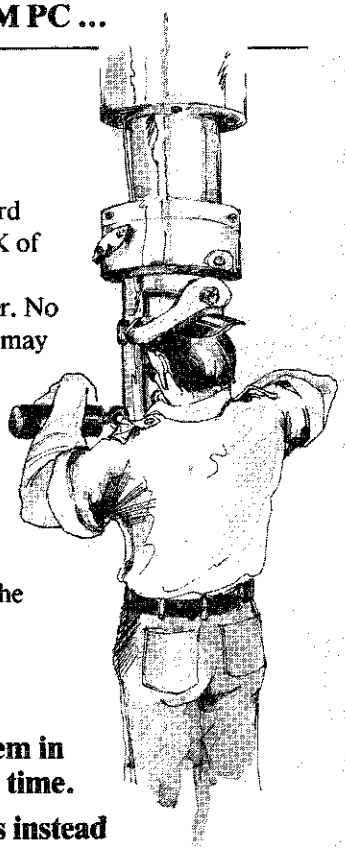
The DEFINT A-Z in Line 500 is standard practice. BASIC runs much faster with integer loop counters and, where possible, integer arithmetic. It is easier to declare the other kinds of variables when required than to keep typing "%" after most of them. The subroutines require Line 505, where TRUE is initialized to -1.

The demonstration program in Lines 1000 through 1565 is quite straightforward and was written only to show how these subroutines may be used in application programs. After signing-on, printing instructions, and laying out the screen, the program asks for a date to be entered. Very deliberately, there is no prompt specifying the input format. We would normally consider this poor practice, but here the intention is to show that the user's normal reaction is probably correct. The format is checked and if it is not acceptable, a message is printed and re-entry required. Similarly, the input is checked for calendar validity. The option of entering a second date is offered, and it is checked the same way if entered. If a second date is not entered the user is asked to enter a number of days. This input is checked to make sure it is an integer value, which may be negative.

The program calls the subroutines as needed and displays the day of the week, the date in alpha format, and the day of the year for both dates (the second being calculated from the number of days difference if not entered). The number of


# Debugging Bugging You?

**Torpedo program crashes and debugging delays with debugging dynamite for the IBM PC ...**



**UP PERISCOPE!**

**First, you install the hardware.**

The hardware's a special memory board that fits in a PC expansion slot. Its 16K of write-protected memory contains Periscope's resident symbolic debugger. No runaway program, however berserk it may be, can touch this memory!

**Then you UP PERISCOPE.**

Use Periscope's push-button break-out switch to interrupt a running program ... even when the system's hung! Periscope supports Assembly, BASIC, C and Pascal. In addition to the usual debugging capabilities, some of Periscope's features are:

**Stop your system in its tracks at any time.**

**Use symbol names instead of addresses.**

**Run a program on one monitor and debug on another.**

**Monitor your program's execution with Periscope's comprehensive breakpoints.**

**Debug memory-resident programs.**

**Put your time to better use.**

The Periscope system is $295. It carries a 30-day money-back guarantee and includes the memory board, remote break-out switch, debugger software, 100-page manual, and quick-reference card. The memory board is warranted for one year. A demonstration disk is $5.00.

System requirements for Periscope are an IBM PC, XT or Compaq, PC-DOS, 64K RAM, 1 disk drive and an 80-column monitor. For MasterCard and Visa orders only, call 800/421-5300 (ext. R96) 24 hours a day. For additional information, call 404/256-3860 from 9 AM to 5 PM Eastern Time.

Get your programs up and running;

**UP PERISCOPE!**

Data Base Decisions / 14 Bonnie Lane /Atlanta, GA | 30328




PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

days between the two dates is calculated and displayed, and then the user is offered the option of repeating the demonstration or quitting.

It is unlikely that any normal use of these routines will even approach precision limits, but we should be aware of them. The algorithms are valid for any Gregorian date through the year 9999 A.D., but long before that we run into precision problems with PC BASIC. The manual states quite clearly that though single-precision numbers are stored with seven digits and seven digits are printed, only six digits are accurate. DAY.NUMBER! is the largest variable we work with, and that reaches 999,999 on September 19, 2737. If we want to be certain of absolute accuracy beyond that date, then we must work with double precision for all the variables and constants involved in the calculations. There is a fixed relationship between what we call the day number and the astronomers' Julian (no relation to the Julian of calendar fame) Day number or JD #. JD #1 begins at noon on January 1, 4713 B.C. The JD #, starting at noon for any date, can be found by adding 1720991 to the day number. If you are an astronomer and want to use this, take care; we are back into the realm of seven digit imprecision.

The calendar subroutines should run not only on the PC and its cousins, but on any machine running BASIC. We have used long variable names and many spaces and line numbers in the routines to make the code easier to read and understand. The routines will take much less space and run faster if they are compressed and lines are combined. If you know what you are doing, it is also possible to rewrite them as user-defined functions, though they are almost incomprehensible in that form.

Now that the cool green screen can keep track of all your dates, you can take down the Playmate of the Month and use the free wall space for something really beautiful, such as a poster-sized Memory Map!

Jehangir Gazdar & Roomy R. Shroff
Palacimo
B. Desai Road
Bombay 400 026 INDIA

References:

1. The World Almanac and Book of Facts. Published annually by Newspaper Enterprise Association Inc., New York.

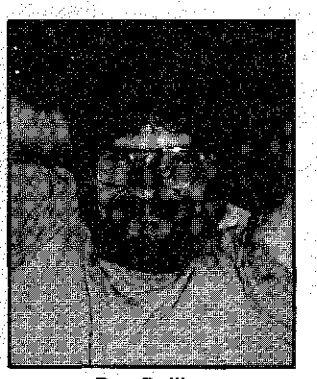
2. HP-67/HP-97 Users' Library Solutions - CALENDARS. Published by Hewlett-Packard Company.

3. Algorithms for RPN Calculators, by John A. Ball Published by John Wiley & Sons, Inc.

Rollinsoft PCsoftware Don Buresh Ron Watson

**PC FIRING LINE / PC UNDERGROUND™**

**Technical Disk Magazine**



**Dan Rollins**

Throw away your paper magazines! Save shelf space, save programming time, with the sizzling ready-to-use source code provided in each powerful bimonthly issue (currently 2 DS/DD disks per issue).

Dan Rollins, captured here relaxing after installing a device driver, has provided PCFL/PCUG with Polymaze, a maze generation program designed to delight children of all ages.

As editor of PC FIRING LINE/PC UNDERGROUND (alias Bill Salkin), I defy you to find the following topics discussed in any magazine—be it paper or disk: hardware maintenance, installing device drivers, critical-error handling, the EXEC function call, random file I/O, different memory models, bicubic splines, "printf" source code, and reducing the flab from IBM Pascal routines. High-sailing columns on Ada, Assembly, BASIC, C, FORTH, FORTRAN, LISP, Pascal, Hardware, and DOS! And NO paper magazine can supply FREEWARE, programmer-oriented DEMOS, and utilities in a ready-to-run format.

**Bill Salkin**

If you have been turned off by disk magazines, then look again. PCFL/PCUG—the magazine of the future—is available today for $12 per single issue (half the price of our competitors' and we contain at least twice their content) or $72 for a one-year (six-issue) subscription.

All funds in U.S. dollars. Foreign countries please send money orders and add $5 (U.S.) airmail for each issue.

Requires an IBM-PC, 128K RAM, and a double-sided disk drive. We ship ONLY DS/DD disks.

**ABComputing**

**P.O. Box 5503, North Hollywood, CA 91616-5503 • (818) 509-9002**

Guy M. Kelly Gary M. Rader Ken Holcombe Microcompatibles



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
100 '*************************************************************************
105 '*                          DATECALC.BAS                                 *
110 '*                                                                       *
115 '*   A program to demonstrate a set of BASIC subroutines which:          *
120 '*                                                                       *
125 '*       1.  Check if a given string is in an acceptable date format.    *
130 '*       2.  Check if the string is a valid calendar date.               *
135 '*       3.  Determine the day of the week of a given date.              *
140 '*       4.  Determine the number of days between two dates.             *
145 '*       5.  Determine the date N days from a given date.                *
150 '*       6.  Convert a date string in the form " MM/DD/YYYY "            *
155 '*           to the "Month DD, YYYY " form.                              *
160 '*       7.  Determine the day of the year for a given date              *
165 '*                                                                       *
166 '*                                                                       *
170 '*       Written in IBM PC / Microsoft BASIC   July 2, 1984              *
175 '*       (C) Copyright 1984 Jehangir Gazdar  &  Roomy R Shroff           *
180 '*                                                                       *
185 '*     ---------------------------------------------------------------   *
190 '*                              LIMITS                                   *
195 '*       Algorithms are valid from the beginning of the Gregorian        *
200 '*       calendar through the year 9999.  Limits of single-precision     *
205 '*       are reached on 09/19/2737 (DAY.NUMBER! exceeds 6 digits).       *
210 '*       Errors are possible after that day unless double-precision      *
215 '*       is used for all variables and constants involved.               *
220 '*     ---------------------------------------------------------------   *
225 '*                                                                       *
230 '*************************************************************************
235 '
500 DEFINT A-Z
505 TRUE = -1
510 '
960 '************************************************************************
970 ' Program to demonstrate the calendar subroutines
980 '************************************************************************
990 '
1000 COLOR 7,0 : KEY OFF : CLS
1005 RTN$ = CHR$(17) + CHR$(196) + CHR$(217)
1010 BAR$ = STRING$(80,205)
1015 CLEAR1$ = SPACE$(240)
1020 '
1025 '--------- set up screen display
1030 '
1035 LOCATE 1,31 : PRINT "CALENDAR  OPERATIONS
1040 LOCATE 3,8 : PRINT "Type a date in the first column, and press [ " RTN$ " ].
1045 PRINT TAB(8) "Either type a date in the second column,
                                or just press [ " RTN$ " ].
1050 PRINT TAB(8) "If you don't type a second date,
                                in the third column type the number
1055 PRINT TAB(8) "of days from the first date for which
                                to calculate the second date.
1060 PRINT TAB(8) "Use a negative number to calculate an earlier date.
1065 PRINT : PRINT BAR$ : LOCATE 21,1 : PRINT BAR$
1070 COLOR 15 : LOCATE 11,5
1075 PRINT TAB(5) "First Date" TAB(35) "Second Date" TAB(68) "# of Days
1080 '
1085 '---------- ask for first date and check it
1090 '
1095 LOCATE 13,5 : LINE INPUT ENTRY$
1100 IF ENTRY$ = "" THEN 1095                    ' must input first date
1105 DTE$ = ENTRY$ : GOSUB 1445
1110 IF NOT VALID THEN LOCATE 13,1 : PRINT STRING$(80, 32); : GOTO 1095
1115 FIRST.DATE$ = DTE$
1120 '
1125 '---------- ask for second date and check it if entered
1130 '
1135 LOCATE 13,35 : LINE INPUT ENTRY$
1140 IF ENTRY$ = "" THEN 1180                    ' get number if no entry
1145 DTE$ = ENTRY$ : GOSUB 1445
1150 IF NOT VALID THEN LOCATE 13,35 : PRINT STRING$(45, 32); : GOTO 1135
1155 SECOND.DATE$ = DTE$
1160 GOTO 1235
1165 '
1170 '---------- ask for a number and check it
1175 '
1180 GOSUB 1555                                  'in case of 2nd date abort!
1185 LOCATE 13,68 : LINE INPUT NUMBER$
1190 GOSUB 1555 : VALID = TRUE : FOR Q = 2 TO LEN(NUMBER$)
1195 IF INSTR("0123456789", MID$(NUMBER$, Q, 1)) = 0 THEN VALID = NOT TRUE
                                                 : Q = LEN(NUMBER$) + 1
1200 NEXT
1205 NUMBER! = VAL(NUMBER$) : IF VALID AND ABS(NUMBER!) >= 1 THEN 1235
1210 LOCATE 23,1 : PRINT CHR$(34) NUMBER$ CHR$(34) " is not a
                                                 number I can use.   Please try again.
1215 LOCATE 13,68 : PRINT STRING$(12,32) ; : GOTO 1185
1220 '
1225 '--------- reverse the print area
1230 '
1235 COLOR 0,7 : LOCATE 15,1
1240 PRINT CLEAR1$ ;
1245 '
1250 '---------- process first date and print results
1255 '
1260 DTE$ = FIRST.DATE$ : GOSUB 2600             'convert to day.date type
1265 GOSUB 2900                                  'get day of the year
1270 LOCATE 15,5 : PRINT DAY$ "," .
1275 LOCATE 16,5 : PRINT ALPHA.DATE$
1280 LOCATE 17,5 : PRINT "is Day #" DOY "of" YEAR
1285 '
1290 IF NUMBER! <> 0 THEN GOSUB 1520             'calculate 2nd date
1295 '
1300 '--------- process second date and print results
1305 '
1310 DTE$ = SECOND.DATE$ : GOSUB 2600            'convert to day.date type
1315 GOSUB 2900                                  'get day of the year
1320 LOCATE 15,35 : PRINT DAY$ ","
1325 LOCATE 16,35 : PRINT ALPHA.DATE$
1330 LOCATE 17,35 : PRINT "is Day #" DOY "of" YEAR
1335 '
1340 '--------- print number.of.days! message
1345 '
1350 IF NUMBER! = 0 THEN GOSUB 2400              'calculate if not given
1355 LOCATE 15,68 : PRINT "They are
1360 LOCATE 16,68 : PRINT ABS(NUMBER.OF.DAYS!)
1365 LOCATE 17,68 : PRINT "day(s) apart.
1370 '
1375 '---------- repeat or end?
1380 '
1385 COLOR 7,0
1390 LOCATE 24,1 : PRINT "Press [Q] to end demonstration, any
                                                 other key to repeat";
1395 Z$ = INKEY$ : IF Z$ = "" THEN 1395
1400 IF INSTR("Qq", Z$) THEN CLS : END
1405 SECOND.DATE$ = "" : NUMBER! = 0 : NUMBER$ = ""
1410 LOCATE 13,1 : PRINT CLEAR1$
1415 LOCATE 16,1 : PRINT CLEAR1$
1420 COLOR 15, 0
1425 GOSUB 1555 : GOTO 1095
1430 '
1435 '---------- date entry checking
1440 '
1445 GOSUB 1555                                  ' clear message area
1450 GOSUB 2000                                  ' check format
1455 IF VALID THEN 1480
1460 BEEP
1465 LOCATE 23,1 : PRINT "Sorry, I cannot understand a date
                                        in the form "CHR$(34) ENTRY$ CHR$(34) "."
1470 PRINT "Please enter it again as Month/Day/Year or as Month-Day-Year." ;
1475 GOTO 1500
1480 GOSUB 2100                                  ' check validity
1485 IF VALID THEN 1500
1490 BEEP
```



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
1495 LOCATE 23,1 : PRINT CHR$(34) ENTRY$ CHR$(34) " is not a
                          valid calendar date!  Please try again." ;
1500 RETURN
1505 '
1510 '--------- calculate second date
1515 '
1520 NUMBER.OF.DAYS! = NUMBER!
1525 GOSUB 2500                          ' dte$ already has first date
1530 SECOND.DATE$ = DTE$                 ' dte$ returns with second date
1535 RETURN
1540 '
1545 '---------- clear message area of screen
1550 '
1555 LOCATE 23,1 : PRINT STRING$(159, 32) ;
1560 RETURN
1565 '
1970 '============================================================================
1975 ' Check DTE$ format =  m[m] / d[d] / yy[yy]   Or    m[m] - d[d] - yy[yy].
1980 ' If valid, DTE$ is changed to standard mm/dd/yyyy form where yy = 19yy;
1985 ' yy = 00 is ambiguous, and is rejected.
1990 '============================================================================
1995 '
2000 VALID = TRUE : FOR Q = 1 TO LEN(DTE$)
2005 IF INSTR("0123456789/-", MID$(DTE$, Q, 1)) = 0 THEN VALID = NOT TRUE : Q = LEN
2010 NEXT : IF NOT VALID THEN 2055                                          (DTE$
2015 FOR Q = 1 TO LEN(DTE$) : IF MID$(DTE$, Q, 1) = "-"
                                  THEN MID$(DTE$, Q, 1) = "/"
2020 NEXT
2025 DELIM1 = INSTR(DTE$, "/") : DELIM2 = INSTR(DELIM1 + 1, DTE$, "/")
2030 IF DELIM1 < 2 OR DELIM1 > 3 OR DELIM2 < 4 OR DELIM2 > 6
                    OR (DELIM2 - DELIM1) > 3 OR (DELIM2 - DELIM1) < 2
                    OR VAL(MID$(DTE$,DELIM2 +1)) < 1
                    OR INSTR(DELIM2+1, DTE$, "/") THEN 2060
2035 IF DELIM2 - DELIM1 = 2 THEN DTE$ = LEFT$(DTE$, DELIM1) + "0"
                                  + MID$(DTE$, DELIM2 - 1)
2040 IF DELIM1 = 2 THEN DTE$ = "0" + DTE$
2045 IF LEN(DTE$) = 8 THEN DTE$ = LEFT$(DTE$, 6) + "19" + MID$(DTE$, 7)
2050 IF LEN(DTE$) <> 10 THEN 2060
2055 RETURN
2060 VALID = NOT TRUE : GOTO 2055
2065 '
2080 '============================================================================
2085 ' Check if DTE$ (in "mm/dd/yyyy" format) is a valid calendar date.
2090 '============================================================================
2095 '
2100 GOSUB 2200 : VALID = NOT TRUE
2105 IF MONTH < 1 OR  MONTH > 12 THEN 2120
2110 IF MONTH = 2 AND DAY = 29 THEN VALID = LEAP.YEAR : GOTO 2120
2115 IF DAY > 0 AND DAY = < VAL (MID$ ("+312831303130313130313031",
                                  MONTH * 2,2)) THEN VALID = TRUE
2120 RETURN
2125 '
2175 '============================================================================
2180 ' Recover month, day, and year values from a date string (DTE$) in
2185 ' the "mm/dd/yyyy" format.
2190 '============================================================================
2195 '
2200 MONTH = VAL(DTE$)
2205 DAY = VAL(MID$(DTE$,4,2))
2210 YEAR = VAL(MID$(DTE$,7))
2215 LEAP.YEAR = (YEAR MOD 4 = 0 AND YEAR MOD 100 > 0) OR YEAR MOD 400 = 0
2220 RETURN
2225 '
2280 '============================================================================
2285 ' Find day of the week of a given date (DTE$).
2290 '============================================================================
2295 '
2300 GOSUB 2700
2305 DOW = DAY.NUMBER! - FIX(DAY.NUMBER! / 7) * 7
2310 DAY$ = MID$("Sunday+++Monday+++Tuesday++WednesdayThursday
                          +Friday+++Saturday+",DOW * 9 + 1,9)
2315 DAY$ = LEFT$(DAY$ + "+",INSTR(DAY$ + "+","+") - 1)
2320 RETURN
2325 '
2375 '============================================================================
2380 ' Determine the number of days between two given dates,
2385 ' FIRST.DATE$ being one, SECOND.DATE$ the other.
2390 '============================================================================
2395 '
2400 DTE$ = FIRST.DATE$ : GOSUB 2700 : FIRST.DAY.NUMBER! = DAY.NUMBER!
2405 DTE$ = SECOND.DATE$ : GOSUB 2700 : SECOND.DAY.NUMBER! = DAY.NUMBER!
2410 NUMBER.OF.DAYS! = ABS(FIRST.DAY.NUMBER! - SECOND.DAY.NUMBER!)
2415 RETURN
2420 '
2475 '============================================================================
2480 ' Determine the date a given number of days (NUMBER.OF.DAYS!)
2485 ' from a given date (DTE$).
2490 '============================================================================
2495 '
2500 GOSUB 2700
2505 DAY.NUMBER! = DAY.NUMBER! + NUMBER.OF.DAYS!
2510 GOSUB 2800
2515 RETURN
2520 '
2570 '============================================================================
2575 ' Convert a date string (DTE$) of the form "mm/dd/yyyy" to a form
2580 ' where the month is spelled out, and another form where the day of
2585 ' week is also spelled out.
2590 '============================================================================
2595 '
2600 GOSUB 2200
2605 MONTH$ = MID$("January++February+March++++April++++May++++++
           June+++++July+++++August+++SeptemberOctober++
           November+December+",MONTH * 9 - 8,9)
2610 MONTH$ = LEFT$(MONTH$ + "+",INSTR(MONTH$ + "+","+") - 1)
2615 ALPHA.DATE$ = MONTH$ + STR$(DAY) + "," + STR$(YEAR)
2620 GOSUB 2300
2625 DAY.DATE$ = DAY$ + ", " + ALPHA.DATE$
2630 RETURN
2635 '
2680 '============================================================================
2685 ' Calculate the day number of a given date (DTE$).
2690 '============================================================================
2695 '
2700 GOSUB 2200
2705 IF MONTH > 2 THEN NMONTH = MONTH + 1 : NYEAR = YEAR :
                       ELSE NMONTH = MONTH + 13 : NYEAR = YEAR - 1
2710 DAY.NUMBER! = DAY + FIX(365.25 * NYEAR) - FIX(.75 *
                       (FIX(NYEAR / 100) - 7)) + FIX(30.6 * NMONTH)
2715 RETURN
2720 '
2780 '============================================================================
2785 ' Calculate the date of a given day number (DAY.NUMBER!).
2790 '============================================================================
2795 '
2800 NYEAR = FIX(DAY.NUMBER! / 365.25)
2805 NMONTH = FIX((DAY.NUMBER! - FIX(365.25 * NYEAR) + FIX(.75 *
                       (FIX(NYEAR / 100) - 7))) / 30.6)
2810 IF NMONTH < 4 THEN NYEAR = NYEAR - 1 : GOTO 2805
2815 MONTH = NMONTH - 1 + 12 * (NMONTH > 13)
2820 YEAR = NYEAR - 1 * (NMONTH > 13)
2825 DAY = DAY.NUMBER! - FIX(365.25 * NYEAR) - FIX(30.6 *
                NMONTH) + FIX(.75 * (FIX(NYEAR / 100) - 7))
2830 DTE$ = RIGHT$("0" + MID$(STR$(MONTH),2),2) + "/" + RIGHT$("0" +
              MID$(STR$(DAY),2),2) + "/" + RIGHT$("000" + MID$(STR$(YEAR),2),4)
2835 RETURN
2840 '
2880 '============================================================================
2885 ' Calculate the day of the year for a given date (DTE$)
2890 '============================================================================
2895 '
2900 GOSUB 2200
2905 DOY = DAY + 31 * (MONTH - 1) - VAL(MID$("003344555667", MONTH, 1)) - 1 *
                                  (MONTH > 2 AND LEAP.YEAR)
2910 RETURN                                                              □
```



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

## MEMORY SEGMENTATION

*by Bill Salkin, Editor/Publisher of PC Firing Line,*
*Copyright 1984 by ABComputing*

One can view the PC's memory as being partitioned into areas, blocks, or "segments" of memory. Segments are not "hard-wired" in any sense; they are a conceptual image of the PC's memory that is needed to develop assembly language programs.

To access a data item a program must create a segment containing that data item. (Otherwise, the data item can not be used by the program.) This emphasizes that segments do not "just exist," nor is the PC memory naturally segmented; the program must create and support this conceptual model.

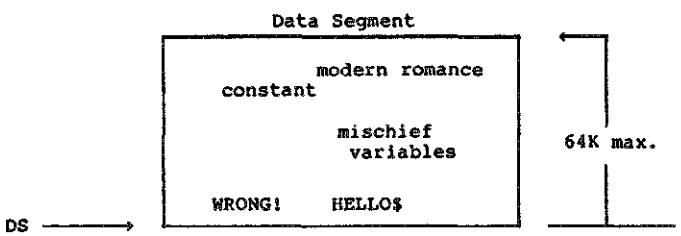
A segment can begin at any location in memory, and segments need not be adjacent. Segments with different names can even occupy the same physical memory address, making them physically identical but logically distinct. The size or amount of data held in a segment varies from 0 bytes to 64K bytes.

We have met the CODE, DATA, and STACK segments in earlier articles. These are areas in memory reserved for the source CODE of our module, the DATA area (for variables and constants), and the STACK, respectively. They are nothing more than holding bins for selected items.

### SEGMENT REGISTERS

Each segment is associated with a 16-bit segment register: the stack segment is associated with the SS register, the data segment with DS, and the code segment with CS. The value in the DS register (when slightly modified, as explained later) is actually the address of the first byte of the data segment. We say that DS points to (symbolized by –> ) the data segment. Similarly the SS register points to the start of the stack segment, and CS points to the code segment.

Study the following "memory snapshot" of a data segment:

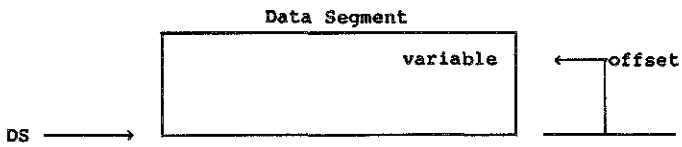



As the snapshot shows, DS points to the start of the data segment, which can hold from 0 bytes to a maximum of 64K bytes of data. In fact, the maximum amount of "anything" in any segment is 64K.

Doesn't the number 64K seem familiar? Advertisements for early compilers for the PC often stated that 64K was the maximum amount of data or code that could be processed. Also, as a segment register is a 16-bit register and 2 16=64K, there is a lot more than coincidence going on here.

We must address (terrible pun!) some further issues before fully explaining this phenomena.

The first issue is to explore how data is accessed in a data segment. Assuming that the DS register points to the start of our data segment, we need an extra "dimension" or way, to reach into the data segment and pinpoint the item we want to process. This extra dimension is called an "offset."




This snapshot indicates that the DS register points to the start of our data segment and the offset value "points the finger" or selects a particular data item. In this case, the DS register and the "offset" value are primed to access "variable."

Being more specific, as "offset" values are generally stored in 16-bit registers, the largest value that an offset can have is 64K (2 to the 16th power). Presto! The magic number. This means that we cannot access more than 64K of data from the start of a given data segment, or more than 64K of code from the start of a code segment, or 64K of anything from any segment register.

In summary, the beginning of a segment is pointed to by the value in a 16-bit segment register (CS, DS, or SS), and the offset from the beginning of a segment is used to access individual data items within the segment. The 16-bit value for an offset limits the amount of data in a segment to 64K. This "offset"approach is also used for the stack and code segments.

### CS AND IP

The offset into the code segment (pointed to by the CS register) is called the instruction pointer (IP). As shown in the



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Debug snatch below, the IP and CS registers were there all the time, just waiting for our attention. We weren't aware of them, though.

```
                                  vv      vv
DS=094F ES=094F SS=0962 CS=095F IP=0003 NV UP DI PL ZR NA PE NC
```

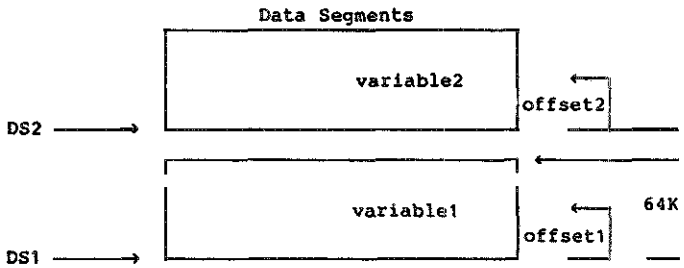
The Instruction Pointer points to a specific location in the code segment. And what is contained at this location? A machine language instruction. After executing this instruction, the value in the IP register is updated to point to the next instruction to execute.

## SMASHING THE 64K BARRIER

Earlier we saw that the maximum offset from a given segment register is 64K. Suppose that our program contains 64K +1 bytes (a programmer-oriented number) of data. How can we access more than 64K of data?

Let's consider a true story. I was telling a friend about a 199-page book (another likely number!) that I had read. To my amazement he jokingly said that he could not count higher than 100 and therefore could not read the entire book! (This situation is equivalent to accessing data outside the range of a segment. In this example, the segment register points to the beginning of the book, and the maximum allowable offset value is 100.) My reply, based upon my knowledge of the 8088 architecture, was "Read to page 100, and then from page 100 read the remaining 99 pages." What could be simpler? This example not only illustrates the concept of segment and offset, but further indicates the manner in which more than 64K of data can be accessed.

Two adjacent 64K data segments are shown below. (Recall that a segment is simply an area in memory; there can be multiple segments used for storing data.) The lower data segment contains the variable "variable1", while the top data segment contains "variable2".




If the DS register (which points to the data segment) contains the value DS1, we may think of DS1 as being the current data segment, and we speak of it as such. With DS1 as the current data segment, offset1 is used to access variable1. We cannot access variable2 using DS1 as our data segment pointer, because variable2 is outside the range of DS1. So we point DS to DS2, which makes DS2 the current data segment. Then "offset2," the offset from DS2, is used to access variable2. (See the analogy between this example and the 199-page book example?) One must "bump" (increase or decrease) the value in a segment register to access information that would require an offset greater than 64K from the original segment register.

This description is correct, but hides many of the implementation details. To access data located anywhere within, say, a 128K block, we must first check if it can be accessed from the current data segment. If it can, then access the data. If it can't, then we "bump" the value in the DS register to point to a segment containing the data item. This approach is more involved than using only one data segment. Tables of segments and offsets may need to be maintained to effectively manage memory accesses. For this reason, one rarely sees (I have never seen) a processor that supports only 128K, or 192K, of data. Having gone to the trouble of exceeding 64K of data, it is little more than an annoyance to support access to the entire memory space. Many compilers went from 64K of data to "full" memory support; there were no intermediate


**Even If You Don't Have The 8087...**

debug 8

from RollinSoft

➪Extends & enhances the power of DEBUG by adding complete support for the 8087 numeric coprocessor.
➪Register values are displayed with 18-digit floating-point accuracy.
➪Status, Tag, and Control words are identified.

***PLUS!***

➪Enter decimal values directly into any 8087 register. Initialize the 8087 with one command.
➪Requires DOS 2.x DEBUG.

***New Commands***

K clear screen, set screen mode, attribute, and border
B binary and decimal conversions
H Hexarithmetic with 6 operations
Y one-step trace of CALL, INT, LOOP and REP. One-line trace display.
X 8087 npX display and modify
* timer-interrupt, FWAIT bugs fixed

*Rush $49.95 to:*
**RollinSoft** PO Box 12147, La Crescenta CA 91214
Created by Dan Rollins




PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

stages. You now know why early compilers were often limited to 64K of data or code.

MEMORY MODULES

There are numerous "memory" models. The model in which the data, code, and stack segments fit into one segment (64K) is called the small model of computation. The code I have presented in my columns, and will continue to present, uses the large model. The large model features separate segments for the code, data, and stack, as shown below.

```
;-------------------------------------------------------------------
DATA    SEGMENT PARA                ;;;; This is the data segment

        DB      'HELLO, WORLD'

DATA    ENDS
;-------------------------------------------------------------------

;-------------------------------------------------------------------
STACK   SEGMENT PARA STACK 'STACK' ;;;; This is the stack segment

        DB      40H DUP('STCK')

STACK   ENDS
;-------------------------------------------------------------------

;-------------------------------------------------------------------
CODE    SEGMENT PARA                ;;;; This is the code segment

ASSUME  CS:CODE, DS:DATA, SS:STACK

;--STANDARD LINKAGE TO DOS

BIGDEAL PROC  FAR           ;DO
        PUSH  DS            ;
        SUB   AX,AX         ;
        PUSH  AX            ;           NOT
        MOV   AX,DATA       ;
        MOV   DS,AX         ;                   ALTER!

;-- RETURN TO DOS

        RET                 ;RETurn to DOS

BIGDEAL ENDP                ;END of Procedure
CODE    ENDS                ;END of code Segment
;-------------------------------------------------------------------

        END   BIGDEAL       ;END of routine BIGDEAL
```

I choose this model for its tutorial value. In the small model, discussion of the various segments is not appropriate because, as they are all crunched into 64K of memory, there is no visible sign of their existence. The large model uses separate segments, which I feel is conceptually clearer.

ESTABLISHING A DATA SEGMENT

The segment shown below, called Hot_Dog, contains data. We want to make this the current data segment in a program. The name Hot_Dog was used in place of DATA to avoid confusion. It emphasizes that the assembler / linker does not recognize a segment as being data, code, or stack merely by the segment name (in this case Hot_Dog). In past columns I had called this segment DATA because immediately upon entering the program I pointed the DS register to this segment. A self-fulfilling prophecy.

```
;-------------------------------------------------------------
Hot_Dog SEGMENT PARA
        DB      'HELLO, WORLD'
Hot_Dog ENDS
;-------------------------------------------------------------
```

If we wish to print the string "HELLO, WORLD" in segment Hot_Dog, we must first point the DS register to the start of this segment. Otherwise, we cannot access this string.

The following instructions in a code segment establish Hot_Dog as the current data segment:

```
ASSUME   CS:CODE, DS:Hot_Dog, SS:STACK
                  ...  ...      ...
MOV    AX,Hot_Dog
MOV    DS,AX
```

The ASSUME statement tells the ASSEMBLER to assume that DS points to the start of the Hot_Dog segment (and that CS points to the code segment, and SS to the stack segment.)

But wait! We have only told the assembler to assume that DS points to Hot_Dog. When the program is executed it must actually place the starting address of the Hot_Dog segment into the DS register. This is accomplished by:

```
MOV    AX,Hot_Dog       ;Move address of Hot_Dog into AX register
MOV    DS,AX            ;Point DS to Hot_Dog
```

In conjunction with the ASSUME statement, these last two statements have established the addressability of Hot_Dog


C

UTILITY LIBRARY

The C UTILITY LIBRARY is a set of 200+ functions designed specifically for the PC software developer. Use of the Library will speed up your development efforts and improve the quality of your work.

- BEST SCREEN HANDLING AVAILABLE
- WINDOW MANAGEMENT, COLOR GRAPHICS
- DOS 2 DIRECTORIES, COMMUNICATIONS
- KEYBOARD, PRINTER, TIME/DATE
- EXECUTE PROGRAMS, BATCH FILES
- STRINGS, BIOS, AND MUCH MORE
- ALL SOURCE INCLUDED—NO ROYALTIES

Available for Microsoft/Lattice $149, Computer Innovations $149, Mark Williams $149, DeSmet $99. Add $3 shipping. N.J. residents add 6% sales tax. Visa, MC, checks—10 days to clear.

Order direct or through your dealer. Dealer/Distributor inquiries welcome.

ESSENTIAL SOFTWARE, INC.
(914) 762-6605
P.O. Box 1003
Maplewood, N.J. 07040




PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

from the DS register, so we may now speak of Hot_Dog as the current data segment.

Again, establishing the addressability of a segment is a two-part process. The first part uses the ASSUME statement for the benefit of the assembler, while the second part, loading the starting address of the segment into DS, is accomplished when the module is run.

MAMMOTH MEMORY

One of the often-touted features of the IBM-PC is its extended memory range — a full megabyte. Wow!

As the segment registers are 16-bit registers, the largest numeric value that can be contained in a segment register is FFFF hex (or 0FFFFH), or 65535 decimal. Using this as the maximum allowable segment address, and recalling that the largest value of an offset is also 64K, it seems that we can not access data whose address is greater than 128K. Maybe all that extra "potential" memory can't be used? But we all know how this story ends, don't we? For one can purchase 576K RAM boards, and no one would do so if a happy ending weren't guaranteed.

Well, then how can a size 10 squeeze into size 8 pants? The trick: the value in a segment register is given a special meaning.

The instruction

```
MOV   AH,7H
```

moves the value of 7H (hex) into the AH register.

The instruction

```
MOV   DS,5H
```

moves the value 5H (5 hex) into the DS register, BUT when the value in DS is used in the calculation of addresses, it is regarded as being 50H. Notice that an extra zero has been appended. Other values are treated similarly:

```
DS  contains  10H     Interpreted as  100H = 256 decimal
DS  contains  100H    Interpreted as  1000H
DS  contains  0ABCH   Interpreted as  0ABC0H
```

Numbers in segment registers should be regarded as having a zero appended to their right side, and, as the above examples show, this multiplies their value by 16. As 64K (the "maximum" value contained in a segment register) * 16 (resulting from shifting values in the segment register) = 1024K, the 8088 can address over 1 megabyte of memory! Pleasant dreams.

This is how the 8088 addresses memory — through deceit! But it works. And this deceit pervades the structure of 8088 assembly language.

SEGMENT:OFFSET ADDRESSING

The address of an item in the range of DS1, at offset of "offset1" from DS1, is symbolized by

```
DS1:offset1
```

The segment value (DS1) is separated from the offset value (offset1) by a colon. The actual address pointed to by DS1:-offset1 is

```
16 * (contents of DS1) + offset1
```

In the following Debug snatch, the leftmost portion shows an address of 959:0001.

```
0959:0001 2BC0 SUB AX,AX
```

This is interpreted as being the actual address 9590 + 0001 = 9591H.

To make this representation even more concrete, we consider the video RAM area, which is in some sense an image of the screen currently displayed on your monitor. Moving a character into video RAM causes this character to be displayed on the screen.

For the monochrome monitor the physical address of the video RAM area is B0000H (that's a B followed by 4 zeroes), and it occupies 4K bytes. To write data directly to video RAM (called memory mapping) we enclose video RAM in a segment starting at B0000H. (Most of this segment is empty as video RAM occupies only 4K of it. It may seem disturbing that we are using so little of this segment, but segment space is not a perishable commodity or in short supply. We create segments as we need them and discard them with equal abandon.)

To make video RAM the start of a segment we place a value in DS so that

```
DS:0   -> B0000H
```

which can be done as follows:

```
MOV DS,0B000H ;notice only three zeroes are used, as a
              ;number in a seg reg. is multiplied by 16.
```

Thus, DS:0 (an offset of zero bytes from DS ) points to the first byte of video RAM (B0000H), while DS:1 points to the next byte (B0001H), DS:2 to the next byte (B0002H), etc.

Let's re-examine the following program presented in an earlier article.



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
;COPYRIGHT 1984 BY ABComputing
;
;
;
;     This Routine displays the string "I have the world on a
;     string"  and returns to DOS.
;
;
;
;-----------------------------------------------------------------
DATA    SEGMENT PARA

STRINGY DB  'I have the world on a string','$'

DATA    ENDS
;-----------------------------------------------------------------

;-----------------------------------------------------------------
STACK   SEGMENT PARA STACK 'STACK'

        DB    40H DUP('STCK')

STACK   ENDS
;-----------------------------------------------------------------

;-----------------------------------------------------------------
CODE    SEGMENT PARA

ASSUME  CS:CODE, DS:DATA, SS:STACK

          ;The ASSUME statement tells the assembler to  assume
          ;that the CS register points to the code segment; DS
          ;to the data segment, and SS to the stack segment.


;--STANDARD LINKAGE TO DOS

DISPLAY PROC  FAR             ;DO
        PUSH  DS              ;
        SUB   AX,AX           ;
        PUSH  AX              ;          NOT

        MOV   AX,DATA         ;
        MOV   DS,AX           ;                    ALTER!


          ;The last two MOV  instructions move the address  of
          ;the segment named DATA  into  the DS  register. The
          ;instruction
          ;
          ;
          ;       MOV   DS,DATA
          ;
          ;
          ;is not allowed by  the assembler, so we  must first
          ;move  the  address  of  this  segment  into  the AX
          ;register, and then move the value in  AX  into  the
          ;DS register.  Extra card-shuffling.
          ;
          ;The segment named DATA  may now be regarded  as our
          ;data segment.

;--Print the String "STRINGY"

        MOV   AH,9            ;request DISPLAY STRING service
        MOV   DX,OFFSET STRINGY  ;make DS:DX -> STRINGY
        INT   21H             ;invoke DISPLAY STRING service


          ;DOS function-call 9 is used to print a string.  The
          ;DOS manual states that we must place a 9 in the  AH
          ;register,  point  DS:DX  to  the  dollar-sign  ($)
          ;terminated string  we want  to display,  and invoke
          ;the function call.
          ;
          ;How do  we make  DS:DX point  to the  string to  be
          ;displayed (STRINGY) ?  Consider the value  already
          ;in DS. DS points to the DATA segment which contains
          ;STRINGY,  so  DS  is  properly  set.  The "offset"
          ;operator determines  the offset  of an  item in the
          ;segment where it is defined.  The instruction
          ;
          ;             MOV   DX,OFFSET STRINGY
          ;
          ;finds the  offset (in  bytes) of  STRINGY, measured
          ;from the start of  DATA, and moves this  value into
          ;the DX register.
          ;
          ;In summary, if DS points to the segment  containing
          ;STRINGY, "pointing DS:DX to STRINGY" is  equivalent
          ;to:
          ;
          ;             MOV   DX,OFFSET STRINGY


;--Return to DOS

        RET                   ;RETurn to DOS

DISPLAY ENDP                  ;END of Procedure
CODE    ENDS                  ;END of code Segment
;-----------------------------------------------------------------

        END   DISPLAY         ;END of routine DISPLAY
```



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

**THE HAMMER**

Software Tools in C

Designed for the programmer with the user in mind.

More than just BIOS/DOS access. , THE HAMMER ALSO provides screen attribute control, Smartmodem control, UNIX-like pattern matching, 123-like command menus, date & time processing routines, easy data entry & verification of dates, numbers, and strings, AND MORE. Super data entry routines give the programmer an easy to use, solid interface with the user.
Requires: IBM-PC/XT, DOS2, CI-C86 Compiler
Other C compilers supported soon.

Routines tried & true. $110 with source and manual.
For ordering or inquiring, call or write now:

O E S SYSTEMS
1906 Brushcliff Road
Pittsburgh, PA 15221
412/243-7365

Looking for the right tool for the job?
**GET THE HAMMER**

**PROFESSIONAL BASIC™**

**JUST BECAME AFFORDABLE**

**$99**

***"marvelous"*** Jerry Pournelle - *BYTE*

**Use 640k (e.g. 250 x 250 array)**
**Dynamic Syntax Checking**
**19 Debugging Windows**
**Run PC BASICA Programs**

**OPTIONAL: 8087/80287 Support — $50**

**Morgan Computing Co., Inc.**
(214) 739-5895
10400 N. Central Expwy., Suite 210
Dallas, TX 75231

SUBROUTINE CALLS

To close this lesson, we return (another pun) to the subject of subroutine calls. As you will doubtlessly recall from previous issues, subroutine calls in assembly language are similar to those in higher-level languages: a subroutine is called by using the CALL instruction, and when the subroutine terminates it RETurns control to the instruction following the CALL instruction in the calling program. (Read that sentence twice before complaining to me!)

What we didn't mention was that there are two kinds of calls: NEAR and FAR.

To create an analogy with the telephone system, a NEAR-subroutine call corresponds to a local-phone call, while a FAR-subroutine call corresponds to a long-distance phone call. Continuing this analogy, as a long-distance phone call is prefixed by an area code, we would expect a FAR-subroutine call to require more information than a NEAR one. This is in fact the case.

If all of the code in a module can fit into a segment, then all calls are termed NEAR calls. (Let me inform you that 64K of assembly language is a BIG program. The PCFL / PCUG main driver, with all of its data, file management, memory management, panels, etc., requires only 9K! Further proof that the large modules created by compilers are full of 'hot air'!) In all likelihood most subroutine calls that you will use will be NEAR calls.

For code size exceeding 64K, multiple code segments are involved and a subroutine may not lie in the same segment as the CALL instruction that invoked it. A FAR call is needed in this case, and, as you may imagine, the "area code" of our long-distance telephone example becomes the address of the code segment containing the called routine.

In summary, if all the code in a module can be squeezed into a segment, then any CALL instruction and the subroutine being called are in the same code segment so a NEAR call is performed. A NEAR call affects only the value of the offset into the current code segment (called the instruction pointer, remember?). After a NEAR call the instruction pointer points to the first instruction of the called subroutine.

In a FAR call, the CALL instruction and the called subroutine lie in different code segments. To fully specify the address of the called subroutine we must provide both a new code segment value (CS) and the offset (IP) into the new code segment.

In the phone system, a long-distance call costs more than a local call. A FAR call also costs more (requires more time to execute) than a NEAR call. But, unlike the phone company's, the rates of FAR calls are not constantly rising.



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
;COPYRIGHT 1984 BY ABComputing
;
;
;     This routine CALLs an Internal Subroutine (SUB).
;
;
;
;-------------------------------------------------------------------


CR   EQU 10                     ;carriage return = ASCII 10
LF   EQU 13                     ;linefeed = ASCII 13


;-------------------------------------------------------------------
DATA      SEGMENT PARA

MSG1      DB    'POSITIVELY, PAUCITY','$'
MSG2      DB    CR,LF,'OF THOUGHT','$'

DATA      ENDS
;-------------------------------------------------------------------

;-------------------------------------------------------------------
STACK     SEGMENT PARA STACK 'STACK'

          DB    40H DUP('STCK')

STACK     ENDS
;-------------------------------------------------------------------

;-------------------------------------------------------------------
CODE      SEGMENT PARA

ASSUME    CS:CODE, DS:DATA, SS:STACK


CALLS     PROC  FAR             ;STANDARD LINKAGE TO DOS
          PUSH  DS              ; DO
          SUB   AX,AX           ;
          PUSH  AX              ;          NOT
          MOV   AX,DATA         ;
          MOV   DS,AX           ;                    ALTER!


;--Subroutine SUB is called  and displays MSG1.  SUB  returns control
;  to routine CALLS, and the next instruction (MOV AH,9) is executed.
;
;  As this module is very tiny, we expect the following call to be  a
;  NEAR call. But, how do we know what type of call it is? Proceed
;  onward.

          CALL  SUB             ;display "POSITIVELY, PAUCITY"


;--Display MSG2   ("OF THOUGHT")

          MOV   AH,9            ;request PRINT STRING service
          MOV   DX,OFFSET MSG2  ;make DS:DX-> MSG2
          INT   21H             ;invoke PRINT STRING service


;--Return to DOS

          RET

;--------------------------- START OF SUBROUTINE "SUB" -----------;
                                                                  ;
        ;The word NEAR  tips the assembler  off!  It should  be a ;
        ;NEAR call, as the  size of the .EXE  module is less than ;
        ;64K, so we indicate this fact.   For a good time, change ;
        ;NEAR to FAR, assemble, and link.                         ;
                                                                  ;
;               ||                                                ;
;               vv                                                ;
SUB       PROC   NEAR                                             ;
                                                                  ;
          MOV   AH,9            ;request PRINT STRING service     ;
          MOV   DX,OFFSET MSG1  ;make DS:DX -> MSG1.              ;
          INT   21H             ;invoke PRINT STRING service      ;
                                                                  ;
                                                                  ;
;--Return to "CALLS" - the routine that called "SUB".            ;
                                                                  ;
                                                                  ;
          RET                   ;RETurn to calling program        ;
                                                                  ;
SUB       ENDP                  ;END of subroutine SUB.           ;
                                                                  ;
                                                                  ;
;--------------------------- END OF SUBROUTINE "SUB" -------------;


CALLS     ENDP                  ;END OF PROCedure CALLS
CODE      ENDS                  ;END of CODE Segment
;-------------------------------------------------------------------

          END   CALLS           ;END of routine CALLS.
```



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Not bad at all. You may wonder, though, where the IP and CS values are. They do not appear in our source code, nor have we (knowingly) reserved any storage for them. The answer lies in a future article, but suffice it to say that these values are saved (PUSHed) and restored (POPped) from the stack. Invisible ink. Egad, it all ties together like a domino theory!

CONCLUSION

We introduced the idea that memory should be viewed as more than just a linear array of bytes. It should be conceived of as being divided into segments where each segment cannot contain more than 64K of information. Memory segmentation proved to be a limiting but powerful feature.

The DS register points to the start of the data segment; SS points to the start of the stack segment, and CS points to the code segment.

Next we introduced the concept of "offset" or displacement into a segment. The address of any item in memory is specified by both a segment value and an offset into that segment. This is indicated in expressions such as DS:DX.

To address the full memory range of the 8088 (over 1 megabyte), we saw that the value in a segment register is given a special interpretation when used in address calculations. That value is multiplied by 16 when an actual address is being determined. Therefore, if DS contains 50H, and DX contains 10H, then

```
DS:DX = 50H:10H
```

represents the actual address

```
500H + 10H
```

Further, we mentioned the difference between NEAR and FAR calls, an idea closely related to memory segmentation.

A future article includes "live" assembly lanuage examples further illustrating these topics. Happy reading.

*The previous article was taken from the disk magazine PC Firing Line from ABComputing, issue#4. PC Firing Line is a technical disk magazine dealing exclusively with programmer related topics as well as supplying a great deal of source code as well as demo programs from various software development support companies. For a copy of this article as well as a full diskette of other programming articles send $12 to:*

*PC Firing Line, issue #4*
*ABComputing*
*P.O. Box 5503*
*North Hollywood, CA 91616*
*(818) 509-9002* □


**YOUR CODE MAY BE WASTING ITS TIME!**
**THE PROFILER™ CAN HELP . . .**

- **Statistical Execution Profiler**
- **Works with any language**
- **Completely configurable**
- **Up to 16 partitions in RAM/ROM**
- **Time critical code optimization**
- **Abnormal code behavior tracking**
- **Graphic presentation of results**
- **Easy to use menu interface**

**THE PROFILER** is a software package which gives you, the programmer, a powerful tool for locating time consuming functions in your code and allows you to performance tune your program. With **THE PROFILER** you can determine where to optimize your code for maximum benefit, then measure the results of your efforts.

Using **THE PROFILER**, you can answer questions like:
Where is my program spending its time?
Why is my program so slow? What is it doing?
Is my progam I/O bound? CPU bound? Are data buffers large enough?
How much improvement did my changes make?

**THE PROFILER** is completely software based and consists of a system resident driver and a monitor program. The memory partitions can range from 1 byte to 1 megabyte in size and can be anywhere in the address space.

**NO ADDITIONAL HARDWARE IS REQUIRED!**

Requires an IBM PC or compatible system with a minimum 64k and one drive.

**THE PROFILER** is available for $125.00 **(New Low Price)** from DWB Associates or ask your software dealer. To order or for more information, call or write DWB Associates. VISA/MC accepted. Dealers welcome.

IBM is a trademark of IBM Corp. MSDOS is a trademark of Microsoft Corp.
**THE PROFILER** is a trademark of DWB Associates.

dwb
Associates
**P.O. Box 5777**
**Beaverton, OR 97006**
**(503)629-9645**

Famous Computer Personalities
Number 512
Stromberg Fernald, Inventor
of the first I/O Port



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

## PASCAL OBJECT CODE SIZE SOLUTION

There seems to be a great deal of interest in our modification of the front end of the Microsoft/IBM Pascal compiler among those Pascal programmers we have so far contacted. By using this revision, it is possible to dramatically reduce the size of the final Pascal object code in those instances where the default allocations in the data segment (stack, heap, and statical) are much larger than necessary.

As a service to PJ's readers who are registered Microsoft/IBM Pascal users, we would be happy to make the relevant information available to those who send us a self-addressed, correctly stamped mailer and diskette. Our address is below.

*Jay Stewart*
*Director of Marketing*
*Pascal Front End Revision*
*Blaise Computing Inc.*
*2034 Blake St.*
*Berkeley, CA 94704*
*(415) 540-5441.*

The following information is a partial explanation of Blaise's Runtime modification.

MS-PASCAL RUNTIME SYSTEM CONTROL

Introduction and Background:

The entry point of every MS-Pascal program is the procedure BEGXQQ. This procedure performs a number of tasks including initialization of global variables, contexts and the runtime memory configuration. The procedure (and other closely related functions) all reside in the file ENTX6SL.ASM distributed with the MS-Pascal Compiler. The routines in ENTX6SL.ASM (all written in Assembler) make up the MS-Pascal Runtime System Control.

One of the most important functions performed by the Runtime Control is to relocate data (constants and static variables) higher in memory creating a gap which can then be used for the Stack and Heap. The system checks the amount of physical memory available on the machine and uses as much as it can, but no more than 64K (after all, all data must reside in a single data segment). The net effect is that even a small program, after Runtime Control executes, will use 64K of memory for its data segment. This fact makes it difficult to use Pascal for small utilities which need to be runtime resident, or front-end programs whose only purpose is to invoke other programs as child processes (using, for example, EXECMM from Pascal TOOLS 2).

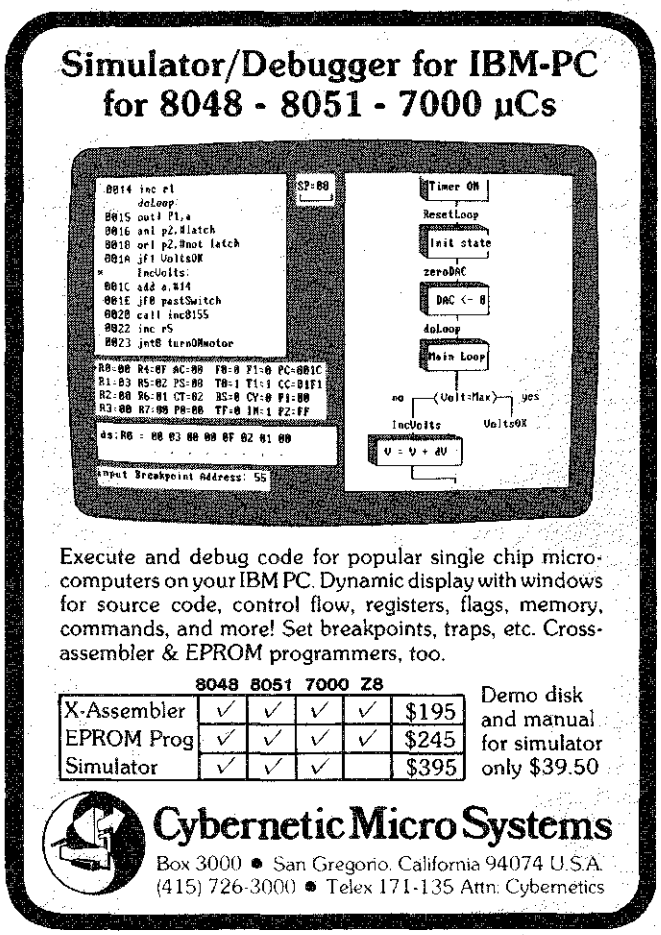
The purpose here is to describe changes which can be made to the Runtime Control System so that the memory use will be less, and to make it easy to utilize the additional memory.

Please note that all comments apply to the MS-Pascal Version 3.11 and later compilers (including 3.20). The IBM Pascal Version 2.0 compiler uses an equivalent Runtime System Control.

CHANGES TO ENTX6SL.ASM

The file ENTXSML.ASM is an alteration of ENTX6SL.ASM which sets the data space size to the number of paragraphs (16-byte blocks) specified in the public variable DATSQQ. The effect is that the Pascal program will have total data limited to DATSQQ (rather than 64K bytes) minus the size of the static variables and constants area. This change can reduce the runtime size of a Pascal program by as much as 55K (even the smallest Pascal program has about 3.5K of static data used by library routines and 512 bytes allocated for the stack).

To specify the data space size, the Pascal application program must declare DATSQQ as a public variable, and initialize its value with a value statement. This ensures that the value of the variable is known when the Runtime System Control executes.



Execute and debug code for popular single chip microcomputers on your IBM PC. Dynamic display with windows for source code, control flow, registers, flags, memory, commands, and more! Set breakpoints, traps, etc. Cross-assembler & EPROM programmers, too.



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

THE MOST EXTENSIVE LIBRARIES ANYWHERE
for the IBM PC, XT

THE GREENLEAF FUNCTIONS

Total Access to the PC

PROFESSIONAL PROGRAMMER'S TOOLS, With Comprehensive Manual, Full Source, Examples, Demos, & Newsletter.

Supports All Major Compilers, All Memory Models, DOS 1.1, 2.0, 2.1

IBM • Honeywell • Control Data • GE • Lotus . . .
Hospitals • Universities • Government • Aerospace . . .
Why not for your application?

Save Months of Painstaking Research

**GENERAL LIBRARY** – What You Would Write If You Had The Time
• 200 functions • DOS 2.0 • Video • String • Printer • Async • Color • Time & Date • Function Keys • Diagnostic • More

#L2 for Lattice or Microsoft, or #C2 for CI C86 ..... $175.00
#D1 for DeSmet C ..... $ 99.00

**ASYNC COMM LIBRARY** – Interrupt Driven Communications Capability
• Polled Mode • Interrupt Mode • Ring Bufferred • Modem Control • CRC • XMODEM Protocol Pieces • More

#L3 for Lattice or Microsoft, of #C2 for CI C86 ..... $160.00
#D3 for DeSmet C ..... $ 85.00

#CC1 Computer Innovations C86 Compiler V2.1 ..... $349.00
#CC2 Lattice C Compiler V2.1 ..... $395.00

(214) 446-8641
• Order Direct or Ask Your Dealer
• MasterCard or VISA Accepted
• Add $7.00 Shipping Per Item
• Specify Compiler and Our Part Number

GREENLEAF SOFTWARE, INC.
2101 HICKORY DRIVE • CARROLLTON, TEXAS 75006

In order to let a Pascal program easily determine its own memory requirements, ENTX6S-L.ASM also has been changed to set a public variable, SIZEQQ, to the number of paragraphs (16-byte groups) of memory required by the program. The quantity can be recovered by simply declaring the variable SIZEQQ as an external word variable in the Pascal application program.

Finally, the Runtime System Control takes control of the interrupt vector 0 (divide by zero), but fails to restore the vector upon program termination. The changes to ensure that the original vector is restored were made to both BEGXQQ and the termination routine ENDXQQ.

Rather than try to detail each change in this document, we refer you to the file ENTXSML.ASM itself. All changes have been documented. Although perhaps sacrificing some efficiency (not much though, because these functions are executed just once), no original code has been deleted. All changes are additions to the code. The original statements are in upper case, all new code is in lower case. Comments which begin with ;* rather than just the semicolon are ours. If a symbol is in upper case, it means it was defined in the original file. Note that to change the heap/stack size, only one statement needs to be altered, and then ENTXSML reassembled.

All changes were made so that implications are isolated to the Runtime System Control. For example, the reason for reporting the program size is so that Pascal code need not change if the Runtime System Control is altered. The Pascal application should, during execution, check the program size (SIZEQQ) if the information is needed.

## IMPLEMENTING THE ALTERED RUNTIME CONTROL

The new Runtime System Control object file, ENTXSML.OBJ, must be used. If the new object file is specified first on the LINK statement command line, the new versions of BEGXQQ (and other routines) will be used, rather than the standard version from the Pascal Library. For example, to link the program INVSML, the LINK statement could be:

```
link invsml+entxsml,invsml,invsml,ctools2/nog;□
```



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

## CORRECTING OVERSIGHTS IN DEBUG.COM

*by Daniel J. Jones, Vice President, Smartware*

I have been a systems programmer for some years, and have found many occasions to use the DEBUG.COM program which is supplied with the IBM PC. I find the program to be helpful in finding bugs in both the PASCAL and ASSEMBLER programs I work with. I also use Debug to observe the execution of new assembler code. Sometimes, when I watch the code as it performs, I discover some code optimization ideas.

Everyone who uses Debug knows that there is a random problem in the trace command. If you have been browsing the PC magazines, you have seen a fix for this trace bug. If you have not, then you are accustomed to the trace command occasionally wandering to parts outside the realm of your program (usually an INT instruction).

While using Debug, I have noticed some oversights in the implementation. The oversights I will mention are not bugs. They are inconveniences. They slow down your productivity. They force the programmer (who is usually a 'hunt and peck' typist) to repeatedly enter numbers and commands. These oversights (and you probably know others) include:

(1) No trace around calls, ints, loops, jumps, and rep primitives
(2) No setting of breakpoints at the current instruction pointer
(3) Limited use of register names in command line
(4) No permanent display of data (repeated use of dump command)

Let's take a closer look at these four limitations, and after discussing them, we'll alter our DEBUG.COM file to correct one of them. To understand these shortcomings of Debug more clearly, we'll use some examples.

### NO TRACE AROUND CALLS, INTS, LOOPS, JUMPS AND REP PRIMITIVES

Imagine that we are tracing through some code and encounter the first limitation of Debug: It does not supply a simple command to trace over a call instruction. To illustrate, suppose we have traced to the following instruction.

```
AX=0010 BX=0012 CX=0004 DX=002F SP=2AE2 BP=0000 SI=2D40 DI=2D41
DS=1041 ES=1041 SS=1041 CS=0841 IP=1F3A NV UP EI PL NZ AC PE NC
0841:1F3A FF976803      CALL    [BX+0368]          DS:037A=2F3D
```

If we don't want to trace through the call, but we want to stop on the next instruction after the call, Debug forces us to set a breakpoint on that next instruction. To set a breakpoint on the next instruction, we need to know its address.

Since the address of the next instruction is not on our screen during tracing, we have two ways to determine the needed address. In one case, we add the number of bytes in the present instruction to the instruction pointer (in the above case 4 1F3A) to determine the instruction address. If we are mentally lazy or don't trust our hex math, the second method is to unassemble the next set of instructions and pick out the address of the next instruction from the screen.

In either case, we must (correctly) type the Debug command g1f3e. Why should we be forced to go through this distraction simply to trace over a call instruction? If Debug must trace through a call, then why doesn't Debug supply a different one-key command to trace OVER a call? Wouldn't it be lovely if we could simply type a one-key command? Perhaps one similar to the trace (T) feature? How about a J command (jump with execution one instruction)?

A "jump one instruction" command would be helpful for tracing around various instructions. For example:

```
AX=3000 BX=0000 CX=2E80 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=0B59 ES=0B59 SS=0B59 CS=0B59 IP=0102 NV UP DI PL NZ NA PO NC
0B59:0102 CD21          INT     21
-


AX=FFF0 BX=FFF0 CX=0000 DX=0080 SP=2AE2 BP=0000 SI=0000 DI=2AFA
DS=0B59 ES=0E4A SS=0B59 CS=0B59 IP=017D NV UP DI PL ZR NA PE NC
0B59:017D 730E          JNB     018D
-


AX=0200 BX=0000 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=0B59 ES=0B59 SS=0B59 CS=0B59 IP=0114 NV UP DI PL ZR NA PE NC
0B59:0114 E2EC          LOOP    0102
-


AX=0000 BX=0000 CX=2E80 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=075F ES=0B59 SS=0B59 CS=0B59 IP=04E6 NV UP DI PL NZ NA PO NC
0B59:04E6 F3            REPZ
0B59:04E7 A6            CMPSB
-
```

### NO SETTING OF BREAKPOINTS AT THE CURRENT INSTRUCTION POINTER

Let's continue our Debug session and explain the second oversight in our list. Suppose we have traced to a particular instruction in our program. Also, suppose we want to set a breakpoint on this instruction. We know our program should eventually reexecute this instruction, and we are interested in seeing how things will look when our program returns. Here is an example of an instruction where we might want to put a breakpoint:

```
AX=3000 BX=0000 CX=2E80 DX=0000 SP=FFF8 BP=0000 SI=0000 DI=0000
DS=0B59 ES=0B59 SS=0B59 CS=0786 IP=0185 NV UP DI NG NZ AC PO CY
0786:0185 FF2EAB0B      JMP     FAR [0BAB]          CS:0BAB=0BFF
-
```

The inexperienced Debug user might enter g185 to set a breakpoint at the next execution of the current instruction.



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The rest of us know that entering g185 will do absolutely nothing. Because of the way Debug sets breakpoints, the command g185 will place an int 3 in address cs:185 and turn execution over to our program, which will execute the instruction at CS:185. The INT 3 at CS:185 will bring us right back to our present instruction without any execution of our program.

So, if we want to set a breakpoint at the next execution of CS:185, we must enter the following two commands:

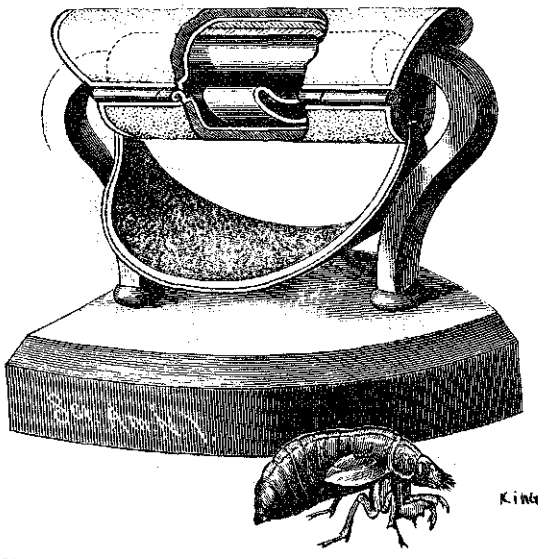
```
-t
-g786:185
```

The T command executes past the instruction at CS:185, and the g786:185 sets a breakpoint for the next execution of the instruction at that address. The segment address (786) is needed because the code segment will most likely change because of our example (a far jump).

What a pain! Wouldn't it be nice if we could just enter g185 or better yet gip (go to IP)? A gip instruction would make setting a breakpoint on the current instruction very easy, using the available contents of the IP register. This brings us to the third oversight we mentioned.

LIMITED USE OF REGISTER NAMES IN THE COMMAND LINE

Debug has a very limited use of register names in the command line. For example, suppose in our debugging task we have traced to the following instruction.




Software Deadline Enhancement Tools: # 3
Certified Device for Ironing Out Bugs

If we want to dump the source string which will be copied by this instruction, we must enter d1530. If we want to display the destination area before the string is copied, we enter des:1050. As we can see, Debug offers some use of the segment register names in the command line, but stops there. Why doesn't Debug allow these?

```
-dsi                    -des:di                 -dbx:dx
-dsp                    -dax:bx                 -dcx:bp
```

On a similar note, why are these not allowed?

```
-ubx                    -ues:ax                 -usp
-hdi si                 -hax 1de4               -hss ds
-gip                    -gbx
-scs:ip fff0 'MASTER'   -sds:ax dx cx           -sss:0 fff0 'COM'
```

The values in these registers are easily available to Debug, so why must we type the values in, rather than the familiar two-character register names? In most cases, the registers contain the addresses of the data we are interested in. The register names don't change, but the values in the registers change all the time. If we could enter the register names instead of the values in the registers, the number of command line typing errors would decrease, and our tracing productivity would increase.

NO PERMANENT DISPLAY OF DATA

The fourth oversight in our list is a major inconvenience. Let's imagine that we are debugging a piece of code which is corrupting its own data segment at address B59:80, but we don't know where the renegade code is.

Our course of action requires setting breakpoints at certain instructions, and dumping the data area which is being corrupted. Each execution to a break point must be followed by a command to dump the affected data area to see if it has been corrupted yet. This requires a maximum amount of concentration and repetition.

Now suppose we could select a line or two of data which we want to be displayed as part of the register display. The following example shows two G commands, with a line of data (0B59:0080) attached to the register display.

```
AX=0B2D BX=0010 CX=0004 DX=002F SP=2AE2 BP=0000 SI=0082 DI=0082
DS=0B59 ES=0B59 SS=0B59 CS=0B59 IP=0217 NV UP EI PL ZR NA PE NC
0B59:0080 01 20 0D 64 65 62 75 67 2E-63 6F 6D 20 0D 00 00..debug
0B59:0217 E82601        CALL    0340                    .com...
-g21a

AX=0B2D BX=0010 CX=0004 DX=002F SP=2AE2 BP=0000  SI=0082  DI=0082
DS=0B59 ES=0B59 SS=0B59 CS=0B59 IP=021A NV UP EI PL NZ NA PE NC
0B59:0080 01 20 0D 64 65 62 75 67 2E-63 6F 6D 20 0D 00 00..debug
0B59:021A E86400        CALL    0281                    .com ...
-g21d

AX=0B0A BX=0010 CX=0004 DX=002F SP=2AE2 BP=0000  SI=2D3F  DI=2D41
DS=0B59 ES=0B59 SS=0B59 CS=0B59 IP=021D NV UP EI PL NZ NA PE NC
0B59:0080 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00........
0B59:021D E8B300        CALL    02D3
-
```



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Surely, we can see the power of the line of data permanently attached to the register display. In our example above, we would suspect the corruption was in the code at B59:281, since the data line changed upon return from the CALL 281 instruction.

These four limitations in the debugger slow down productivity. If an entered number is mis-keyed, the result can be either harmlessly distracting, as in showing the wrong data, or genuinely maddening, as in losing control of the program being debugged.

These oversights are extremely frustrating to programmers who regularly use Debug. They require too much typing, which invites command line errors.

Many programmers are probably avoiding the use of Debug except as a last resort because of these limitations, and others which have not been mentioned.

## ADDING THE J COMMAND TO DEBUG.COM

After defining these oversights in Debug, let me show you how I modified DEBUG.COM to overcome one of them. I will explain how the J command was added to DEBUG.COM. The J command, as you will recall, is my answer to Debug's first limitation. The J command will perform the same as a G command to the next instruction.

First I divided the project into two parts. The first task was to create the code which could perform the J command. The second task was to add the code to the existing DEBUG.COM file.

## TASK 1: CREATING THE J COMMAND

To create the code for the J command, I used Debug to watch the execution of Debug. I soon found limitations when attempting this. For example, debugging a G command or T command will reset soft vectors, causing loss of control. However, after a few hours of observing Debug code executing, I eventually found the information I needed.

What I wanted to do was change the J command (in the command line) into a G command with the address of the next instruction, and turn execution of the G command over to Debug. This would allow me to simply type a J followed by a return, and Debug would actually perform as if I had typed a G followed by the address of the next instruction and a return.

I wanted to use as much of the existing Debug code as possible, so I looked for the following items in Debug code:

(1) Command line address
(2) Instruction length computation
(3) Address of current instruction
(4) Address of command line execution

With these I could supply the command line with a G, compute the length of the current instruction, add it to the address of the current instruction, supply this computed address to the command line following the G, and jump to the address where Debug executes the command line.

I had difficulty finding an instruction length computation in Debug's code. Finally, when I was tracing through Debug's R instruction, I found the call which displays the current instruction, and an address which will contain the location of the next instruction after this call is executed.

So I didn't need an instruction length computation at all. I could call the current instruction display to get the address of the next instruction. The only problem was I would have to disable output to the screen before the call to display the current instruction, and re-enable the screen output after returning.

The list of pieces I needed from Debug looked like this:

(1) Address to disable/enable screen output.
(2) Address where current instruction address is stored.
(3) Address to call to update current instruction address to the next instruction address.
(4) Setup requirements before call in item 3.
(5) Address of command line.
(6) Address of command line execution.

And the course of actions to take:

(1) Disable screen output
(2) Make dummy call to display current instruction
(3) Make note of address of next instruction
(4) Re-enable screen output
(5) Set up command line with G and noted address (hex)
(6) Jump to command line execution.

When I was tracing through version 2.0 of Debug, I found address CS:348 to be responsible for screen output. It held an INT 21. To disable the screen output, I placed two NOP instructions there. To re-enable it, I replaced the INT 21 instruction.

Debug uses location CS:2CEE to store the address of the current instruction. After a call to CS:F82, you would find the address of the next instruction at CS:2CEE (the target of the J instruction). I found one of Debug's calls to F82 beginning at CS:6D9 and ending at CS:6F4. I used this code in my J command p which restores the original contents.



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

I found the command line address at CS:2D3F, while the execution of the command line starts at CS:222. The command line that is eventual used for execution must have been previously converted to upper case alphabetic characters, so I had to insure that I planted an upper case G in the command line.

With all the addresses finally collected, I then wrote a small assembler procedure which would perform the J command. The J command code will be entered via a call, and exit with an unconditional jump to Debug's command line execution code, so the first instruction must remove the unneeded return address. Presented here is my J command code.

---

J COMMAND CODE

```
        pop     ax                      ; remove unneeded ret address

        mov     ds:348h,9090h           ; temporarily disable output

        mov     ax,ds:2AFAh             ; set up dummy display to get G addr
        mov     ds:2CEEh,ax
        push    ax
        mov     ax,ds:2AF8h
        mov     ds:2CF0h,ax
        push    ax
        mov     ds:2D6Fh,0FFFFh
        mov     bx,0F82h
        call    bx                      ; dummy unassemble portion of reg
        mov     ax,ds:2CEEh             ; get address to plant in G command
        pop     ds:2CF0h                ; restore current instr addresses
        pop     ds:2CEEh

        mov     ds:348h,21CD            ; re-enable screen output
                                        ; plant G command
        mov     si,2D3Fh                ; command line addr
        mov     byte ptr [si],'G'       ; plant the G
        mov     byte ptr [si + 5],0dh   ; plant the carriage return
        lea     di,[si + 1]             ; plant the address
        call    ax_to_comline

        mov     bx,222h                 ; jump to command execute
        jmp     bx                      ; expects SI is addr of command line

ax_to_comline:                          ; convert AX to hex string in [DI]
        xchg    ah,al                   ; convert ah first
        call    al_to_comline
        mov     al,ah                   ; then al

al_to_comline:
        push    ax                      ; save al low half
        mov     cl,4
        shr     al,cl                   ; do high half first
        call    low_al_to_comline
        pop     ax                      ; recall low half
        and     al,0fh

low_al_to_comline:
        add     al,90h                  ; convert to hex, put in [di]
        daa
        adc     al,40h
        daa
        stosb                           ; copy to [di] (command line)
        ret
```



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

TASK 2: ADDING THE J COMMAND CODE TO DEBUG

Once the J command code was created, I had to modify the existing DEBUG.COM file. I located the following items in the Debug program.

(1) The code which decides where to load the target program
(2) An address in Debug where I could add my J command code
(3) The call table address which needed the address for my J command code

Debug loads the target program just after itself. Since I wanted to add my code just after Debug's code, to keep the target program from loading on top of my new code I would have to modify the Debug's default address. I found CS:139 to be the address Debug uses to calculate where to load the target program.

Next I decided where I could add my code. Since the trace bug fix that others have already installed starts at CS:2F18, and ends at CS:2F2C, I decided to start my code at CS:2F2D. This would make the modification compatible with DEBUG.COM files which have been previous altered.

Also, since I was going to add a J command to Debug, I had to alter the command call table. I wanted the J entry in the table to call my new code. I found the address for the J command entry in the table at CS:37A.

I had the information I needed, so I used Debug to modify the DEBUG.COM file. Presented here is the screen output of Debug, for patching in the J command.

---

```
>debug debug.com
-e139

0B59:0139  17.90

-e37a

0B59:037A  88.2d   05.2f

-e 2f2d

0B59:2F2D  00.58   75.c7   03.06
0B59:2F30  E9.48   11.03   00.90   8B.90   86.a1   F8.fa   FE.2a   89.a3
0B59:2F38  06.ee   42.2c   0A.50   C7.a1   86.f8   F8.2a   FE.a3   00.f0
0B59:2F40  00.2c   00.50   00.c7   00.06   00.0f   00.2d   00.ff   00.ff
0B59:2F48  00.bb   00.82   00.0f   00.ff   00.d3   00.a1   00.ee   00.2c
0B59:2F50  00.8f   00.06   00.f0   00.2c   00.8f   00.06   00.ee   00.2c
0B59:2F58  00.c7   00.06   00.48   00.03   00.cd   00.21   00.be   00.3f
0B59:2F60  00.2d   00.c6   00.04   00.47   00.c6   00.44   00.05   00.0d
0B59:2F68  00.8d   00.7c   00.01   00.e8   00.05   00.00   00.bb   00.22
0B59:2F70  00.02   00.ff   00.e3   00.86   00.e0   00.e8   00.02   00.00
0B59:2F78  00.8a   00.c4   00.50   00.b1   00.04   00.d2   00.e8   02.e8
0B59:2F80  FF.03   E3.00   86.58   E0.24   E8.0f   02.04   00.90   8A.27
0B59:2F88  C4.14   50.40   B1.27   04.aa   D2.c3
-rcx

CX 2E80
:2e8d

-n smartbug.com

-w

Writing 2E8D bytes
-q
```



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The previous Debug screen output shows the alteration of Debug addresses to make the J command work. CX must be increased to 2e8d to write out the new slightly larger Debug file. The name command is optional. If you want the modified file to be named DEBUG.COM, you don't need the name command. If you don't use the name command, it will overwrite your DEBUG.COM.

If you have the 2.0 or 2.1 DEBUG.COM for the IBM PC, you can add the J command to your file by following the steps shown above. When you enter the code at 2F2D, the bytes you are modifying do not exist in Debug, so don't be surprised if they are not the same as those in my example. You are entering code into random memory which will be written out to the new Debug file.

After you have modified your Debug, try the J command. If it doesn't work, inspect the code to see if you have entered the numbers correctly. Try the unassemble command at address 2f2d, and see if the code looks similar to the assembler procedure I wrote for the J command.

When you use the J command, use it to jump over whatever instruction you like, but keep this in mind. The J command is exactly like a G command to the next instruction. If program execution does not go to the next instruction, then you will lose control. Also, if you use the J command to jump over a call instruction and you lose control, then the call did not return correctly.

Here, at Smartware, we have become spoiled with the J command, and would be very unhappy without it. If you add the J command to your Debug, you will become spoiled too, and wonder why it wasn't already available.

I have presented a brief history of my attempt to correct one oversight I found in DEBUG.COM. I suggest that programming tools need not be taken as is. If an improvement would be helpful, perhaps a little investigation is in order to see if the program can be modified to perform in more helpful ways. In the case of the Debug shortcomings, I have modified DEBUG.COM to correct them all, and now have a more valuable programming tool. If you take the time to search for a way to correct the oversights I have mentioned, you too will enjoy a more useful debugger.

If you don't have the time or patience required to install your own solutions to Debug problems, but would like a more useful debugger, then we invite you to send for our Debug enhancer. We can't send you our SMARTBUG.COM version because it includes copyrighted Debug code, but we can send you our conversion program. Our Debug enhancer reads in your DEBUG.COM program, adds all our new features, and creates an updated version for you. You just need to tell the enhancer program which disk drive your DEBUG.COM file is on, and what you want to name your new copy. The enhancer is available for DOS 1.1, 2.0, or 2.1, and is compatible with Debugs that have the trace bug already fixed. In fact, the trace patch will be added if it is not already there.

If you wish to receive a copy of our Debug enhancer, give us a call or send $25 (we pay shipping and handling, outside the continental US add $3) to :

*SMARTWARE*
*Debug Enhancer*
*1965 Manzanita St.*
*Klamath Falls, OR 97601*
*(503) 883-3278*

Be sure to indicate which DOS version of Debug you have (1.1, 2.0 or 2.1). You don't need to send us your DEBUG.COM, just a formatted blank diskette. □


**OPT-TECH SORT™**

**SORT/MERGE program for IBM-PC & XT**

**Now also sorts dBASE II files!**

- Written in assembly language for **high performance**
  Example: 4,000 records of 128 bytes sorted to give key & pointer file in 30 seconds. **COMPARE!**
- Sort ascending or descending on up to nine fields
- Ten input files may be sorted or merged at one time
- Handles variable and fixed length records
- Supports all common data types
- Filesize limited only by your disk space
- Dynamically allocates memory and work files
- Output file can be full records, keys or pointers
- Can be run from keyboard or as a batch command
- Can be called as a subroutine to many languages
- Easy to use, includes on-line help feature
- Full documentation — sized like your PC manuals
- **$99** —VISA, M/C, Check, Money Order, COD, or PO
  Quantity discounts and OEM licensing available

To order or to receive additional information write or call:

**OPT-TECH DATA PROCESSING**
P.O. Box 2167 Humble, Texas 77347
(713) 454-7428
Requires DOS, 64K and One Disk Drive




PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

## dBASE II TO dBASE III?

*by Ronald Moak*

A couple months ago I was happily developing an applications program in dBase II. Rumors abounded about the new version of dBase that was soon to be on the market. At the time no thought was given to halting the project until the new version was released. When the program finally was available the press releases promised much, but then again how often, in this industry, do press releases come close to matching program performances? After reading a couple of reviews describing the new features my interest was beginning to peak. At the time I felt that if I did not develop my current application in dBase III, it at least held potential for any new programs, or a later rewrite.

As you can probably guess I bought dBase III. In part, the decision was based on pure logic (new features and all that). The other part was based on pure emotion. After a year of working with dBase II on and off, I had mixed feelings about using a new, untried program. I did not relish the idea of sweating through numerous hours uncovering bugs. However, I am still overwhelmed with the ease of developing programs in dBase II that would take many hours longer in any other language, and dBase III promised to shorten the process even more. Once the emotion of having the latest and greatest won out, it was not hard convincing myself of the logic of the purchase.

Since acquiring dBase III, I have been asked by several people if they should upgrade to the new version. My response to that question is, "It depends." When faced with the decision to upgrade, a number of questions should be answered. If you are a "gottahave" person, a person that has to have the latest new program on the market regardless of the need, there is no decision to be made (except maybe how to pay for it). If you are like most of us, there needs to be a little more meat behind the sizzle of the advertisement.

If your current use of dBase II is as a base for running an application program developed by someone else, the answer is probably not. dBase III cannot necessarily execute a dBase II program. It depends on the commands used in the command file. Several programs that I wrote in dBase II worked in dBase III without conversion; however, the programs were limited to a single data file and were very simple in nature. All dBase II data files need to be converted to dBase III format before they can be read by a dBase III program.

If you are running a system with less than 256K memory and single-sided disk drives, dBase III is out of the question until you buy more memory and larger drives. Also, if you are running dBase II on a CP/M based 8-bit machine, the answer is a definite no. dBase III requires a 16-bit based machine to work.

Answers to the rest of the technical questions are not so clear cut. For example, if you are writing programs in dBase III, how much storage are you going to require for your program and associated data? dBase III will run on a single disk drive; however, storage for data and programs is severely limited. The overlay for dBase III requires 144K of storage, and must be present on the program disk. On a two disk system this leaves about 216K of programs and 360K of data, assuming separate disks for programs and data. A fairly complex program could easily require a hard disk for storage.

Unlike dBase II, dBase III is copy protected. You may port the program over to the hard disk but you must still have the system disk in drive A when you boot the program. Once the program is running you can remove the system disk. However, I understand that there is at least one program that allows you to bypass the protection scheme and save your system disks as backups.

Once you have determined you have the hardware requirements to run dBase III, the main decision boils down to whether the new capabilities are worth the investment in time and money to learn the new commands and rewrite the

Software Deadline Enhancement Tools: #5
The Programmer's Brilliant Flash Inducement Aid



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

old programs. Because dBase III is a complex program and I have yet to master all of the intricacies, I will not attempt to show you all of the features that will solve your programming problems. I will try to cover some of the reasons I decided to convert, and some of the reasons I am glad I converted.

One of the new features of dBase III that attracted my attention (aside from the increase in open files and record sizes) was the addition of the date type in records. dBase III allows both memory variables and record fields to be defined as a date type. When two dates are subtracted from each other, the resulting figure is the number of days between the two dates. At the time I learned this I was struggling to design an accounts receivable package that charges interest on accounts over 30 days old and breaks down the receivables into 30, 60, and 90 plus categories. dBase III allowed me to condense a very involved section of code into the following lines:

```
today_date = date()
use invoice index invdate
do while .not. eof()
   do case
      case today_date - invdate > 90
         temp_ar90 = temp_ar90 + inv_amt
      case today_date - invdate > 60
         temp_ar60 = temp_ar60 + inv_amt
      case today_date - invdate > 30
         temp_ar30 = temp_ar30 + inv_amt
      otherwise
         temp_arcur = temp_arcur + inv_amt
    endcase
    skip
enddo
temp_int = (temp_ar90 + temp_ar60 + temp_ar30) *.015
```

Another inducement to purchasing dBase III was the ability to have up to ten files open at a time. I have one input program that opens and closes up to eight files depending upon the information entered. dBase III allowed all of the files to be opened at the beginning of the program; then individual files could be selected as desired. Under dBase II, constantly opening a file, retrieving some information, and then closing the file slowed the process of data entry.

Over the last year I have generated numerous reports using dBase II but I have yet to use the report generator. This is primarily due to the fact that the reports I needed required two or more files to generate. The dBase III report generator has a better interface but is still limited to reporting one file at a time.

dBase III has provided a couple of commands to help make the generation of reports easier. SET RELATION TO <key> INTO <alias> allows me to set a relation between two files based upon a key. When I bump a file pointer in one file, the secondary file is automatically incremented to the correct record. This allows me to access data automatically from a second file without having to stop, search the second file, extract the information and then continue processing the primary file.

The other command I use in generating reports is SET FILTER TO <condition>. With this command I can set a filter on the primary file. All other commands I execute that contain a condition will be executed according to the conditions of the preset filter.

These two commands are not limited to the generation of reports. In fact, the set relation command alone is proclaimed by many to be worth the switch between II and III.

One of the problems of developing a large program in dBase II is the number of small files which seem to accumulate. These files have been set up as subroutines that are frequently called from the main programs. Using multiple small files helps to modularize your code; however, it slows down processing because of the need to open a file each time you branch to a subroutine. dBase III has helped helped me by allowing separate procedures to be set up in a separate file. I can even pass parameters to a routine and receive variables back. Variables used in the procedure are local to that procedure. This allows variables in the procedure to have the same names as variables in the program without disturbing the contents of the program variables. Up to 32 procedures may be stored in a single file; however, only one procedure file may be active at a time. Using standard procedures to handle file searches and error messages saves having to code the routines in every program, or having to execute a separate file each time I choose to use a routine.

In the following program segment, the user is prompted to enter a customer number. The program enters an infinite loop that is exited when the correct customer number is entered. Once the number is entered, the program branches to a procedure and passes three variables. The first variable is a string that contains the customer number. The second variable contains the error message to be displayed if the customer is not found. The final variable contains a return code that indicates the success of the search. Within the look-up procedure, a second procedure is called. This procedure displays the standard error message if the customer is not found.

As you can see, the look-up procedure is generic, and may be used on any file that is opened with an index. This routine is called numerous times from many programs to look up or verify the presence of data. By maintaining and improving the procedure within one file, I am relieved of the necessity of changing the code in every program that calls the procedure.



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
Program file ------------ main.prg

set procedure to procfile
use customer index customer
@ 4,10 say "Customer No. "
do while .t.
   temp_cust = 0
   @ 4,23 get temp_cust picture '999'
   read
   clear gets
   do lookup with str(temp_cust,3), 'invalid customer',ok
   if ok
      exit
   endif
enddo


Procedure file ---------- procfile.prg

procedure lookup
   parameters search, err_mes, ok
   find &search
   if eof()
      ok = .f.
      if err_mes <> ' '
         do error with err_mes
      endif
   else
      ok = .t.
   endif
return

procedure error
   parameters err_mes
   set confirm on
   cmd = ' '
   @ 24,1 say err_mes + space(20) + '<cr> to continue ' get cmd
   read
   clear gets
   @ 24,1 clear
   set confirm off
return
```

ONE KEY IS WORTH A 1000 LINES

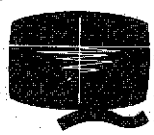
TALL SCREEN™ extends your screen at the touch of a key

PAGE BACK — **PAGE BACK** and **EDIT** up to 1000 lines of previous output. **RECALL** previous DOS commands or previous paths with a single keystroke.

RECALL — **EDIT** text anywhere on the screen. **FILE**, **PRINT**, or **FREEZE** windows on the screen.

EDIT — Run programs unattended, review output at your convenience. Create customized keystroke profiles as in the IBM Personal Editor.

FREEZE — TALL SCREEN is a powerful tool for the PC user who values time and control.



$49.95 INTRODUCTORY PRICE

For orders or information see your local dealer or call **1-800-522-0290**

QUALITAS™ QUALITY PROGRAMMING TOOLS

For the IBM PC®, PC Jr, XT, AT, or compatibles. Compatible with PCDOS 2.00, 2.10, 3.00 and 3.10. Compatible with most mouse systems and joysticks. Works with PROKEY™ and SIDEKICK™. Next day shipping on VISA/MC orders. Price includes all domestic shipping and handling. No extra charge for VISA/MC. MD & NM residents add sales tax.

QUALITAS, INC., P.O. BOX 3AK UPB, LAS CRUCES, NM 88003 505-522-0290




PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The commands that I have presented here are just a few of the additional commands that have been provided to dBase III. Some of the new commands, such as the ones I described above, I use constantly. Others I will probably use over time, as I begin to explore dBase III more deeply.

The addition of new commands is a major reason to consider dBase III but there are still other issues to be considered. Unlike it's predecessor, dBase III allows a link with DOS. If you have sufficient memory in your system, you may execute programs in DOS without leaving dBase III. I have 640K of memory in my system so I set aside 384K for an operating system, and the rest for a ram drive. While 384K seems like a lot of system memory, it allows me the ability to use my program editor and edit several files simultaneously without leaving dBase III. I have set up the main menu of the program I am developing with an option to edit a program module. This way I can test a section of code, and if I'm not pleased, execute the editor, modify the code, leave the editor and end up back at the program menu. There have been many times when I have spent three to four hours working in dBase III with the editor, or performing other DOS commands, without ever reloading dBase III after the system was booted.

Among the other things to consider are human touches built into dBase III. For example, when sorting a large file, if insufficient space is available, the system will halt and allow you the option of deleting files and continuing, or aborting the operation. dBase III provides on-screen menus of key options for many of the operations like editing or browsing a database. I can not count the times I had to stop and look up a control character sequence that I had forgotten in dBase II.

If you are new to dBase and are deciding on whether to purchase dBase II or dBase III, my inclination would be to pay the extra and go with dBase III despite the possibility of bugs. The power offered in dBase II was probably far greater than what the average dBase II user would ever use. The same is true for dBase III; however, built-in mechanisms will allow the average dBase III user to come up to speed a lot faster than dBase II. The assist command provides the new user the ability to navigate through the maze of dBase III commands with the aid of menus. On-screen help is far superior that which was available to users of dBase II.

One of the problems with giving up dBase II is the loss of many of the secondary programs provided to support dBase II. For the most part these programs will not work dBase III. Fortunately, most problems that were solved with other programs have been solved with dBase III. Also included in dBase III is a program generator that may be used to create edit screens.

The last consideration I looked at was to the future of dBase III. Admittedly, this is purely a speculative view of what direction dBase III will take over the next few years. Prime argument for questioning the future of dBase III is due to the loss of a large market of potential purchases. Any programs written in dBase III will lose the 8-bit CP/M market.

As pure conjecture I am projecting the porting of dBase III to other powerful machines and operating systems (e.g., Macintosh, Unix). One of the main supporting reasons for this projection is the fact that dBase III was written in C. No company the size of Ashton-Tate can afford to ignore the the huge market potential that is non-IBM. Portability is the name of the game for software developers; hence the increasing use of C in developing systems programs.

Looking again into the crystal ball one may see a multi-user version of dBase III. After all, there is a multi-user dBase II, so why not dBase III? Runtime version is also probable in the near future, as is a better graphics interface (at least from a third party company).

Should you move up to dBase III? Maybe, then again maybe not. If you can, take a look at it, read the documentation, find someone you know who has it and try it out. dBase III offers a lot of amelioration over dBase II, but is yet to be fully tested. If all goes well, the future looks bright, but someone probably said that about VisiCalc at one time. □


**Attn MS-DOS and PC-DOS users:**

**$25 DEBUG ENHANCER**

Enhance the capabilities of your debug program and experience easier, more productive debugging.

**Use enhancements like**

- New jump command
- Set breakpoint at current instruction
- Use register names in command line
- Attach data line to register display
- Trace bug fix (IBM)
- Hex/decimal conversions

**And enjoy debugging with**

- One-key trace for int, call, rep & loop
- Less keyboard input and repetition
- Less mental math
- Fewer command line entry errors
- No machine hang when tracing

**To order call (503) 883-3278**
or write SMARTWARE
1965 Manzanita Street
Klamath Falls, OR 97601

**VISA** or **MASTER CARD** Accepted. Mailed orders must indicate DOS version number (1.1, 2.0 or 2.1) and machine type. Extra charge for C.O.D. and orders outside the continental U.S. MS-DOS is a registered trademark of Microsoft Inc. PC-DOS is a registered trademark of International Business Machines Inc.




PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

## THINKTANK

*a review by Tom Lichty*

PROGRAM DESCRIPTION

There are only a few programs available today that can do so much and require so little. Learn 'em in half an hour; save 50 hours in the next year using 'em. There aren't many. pfs File. Bank Street Writer. MacPaint, even.

And ThinkTank.

If you're the kind of person who makes lists of everything from What To Do Today (even if it's Sunday) to a presentation before the city council, then you're ripe fruit for ThinkTank. I'd no more write an outline without ThinkTank than I'd take a shower without soap.

Here's the idea:

Taking A Shower
    Enter the bathroom
        If door is closed
            Knock
                Request entry (if appropriate)
                Wait (if necessary)
        If door is open
            Enter bathroom
            Turn on light
            Close door behind you
    Prepare the bathroom
        Position necessary materials
            Towel
            Bath mat
            Wash cloth
            Soap
            Shampoo
        Adjust water temperature
        Turn on exhaust fan
        ...etc.

...an outline, breaking an event (or organization, or plan) into its representative parts. Most of us plan important things in this way.

ThinkTank automates this process, automatically indenting and keeping track of what goes where, even if insertions or deletions are made. Moreover, ThinkTank allows you to work with as much detail as you want. Position the cursor on "Position the necessary materials" (above) and press the MINUS key. Here's what happens:

Prepare the bathroom
    Position necessary materials
        Towel
        Bath mat
        Wash cloth
        Soap
        Shampoo
    Adjust water temperature
Turn on exhaust fan

...becomes:

Prepare the bathroom
    Position necessary materials
    Adjust water temperature
    Turn on exhaust fan

...the detail collapses. The PLUS key retrieves the detail. Instantly. All detail can be collapsed into its respective header at the touch of a key. If the cursor were to be positioned on the first line, "Taking A Shower" (above), ALL detail would collapse into that line:

Taking A Shower.

...just the first line. All detail is subordinate to this first line; all detail collapsed into it. You can display as much or as little detail as necessary.

There's more: suppose "Adjust Water Temperature" requires an explanation in narrative form. Something like:

Prepare the bathroom
    Position necessary materials
    Adjust water temperature
        Inside the shower itself, you will find two knobs, labelled "HOT" and "COLD". Turn both knobs counterclockwise until they stop. This should start water flowing from the shower head. After a minute has passed and the water temperature has stabilized, adjust the temperature by lowering the appropriate temperature in the mix of hot and cold. Do so by turning the appropriate knob clockwise until the proper temperature is reached. Gauge the temperature by holding your hand under the shower spray.
    Turn on exhaust fan



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

...ThinkTank gives you nearly complete control over the narrative section, providing the most common functions of any screen-oriented word processor. You'll have to change modes to write such a narrative (ThinkTank calls this the "Paragraph Editor"), and the outline section won't be available until you switch modes back again, but the Paragraph Editor provides MOVE, COPY, MERGE, and DELETE — even SEARCH and REPLACE — functions like any good word processor.

Essentially, that's all ThinkTank does. That's its essential elegance: it performs a simple task in a straightforward manner. Learning the program will take no more than half an hour. Furthermore, ThinkTank is a menu-or-command program: press F10 and a menu of ThinkTank's functions appears. Move the cursor among them (the cursor travels over the selections much like Lotus 1-2-3, exposing brief explanatory lines with each new command) and select the one you want with the RETURN key. Alternatively, use the function or ALT-combination keys to select commands and bypass the menu: have the best of both worlds.

PRINTING

Nothing comes up short here. ThinkTank provides seventeen print formatting commands: margins, indentations, headers and footers, setup strings — they're all under your control. Want to output to a word processor or a spreadsheet? ThinkTank allows printing to disk, in standard DOS text files. Want a formal outline format? How's this:

1.2 Prepare the bathroom
- 1.2.1 Position necessary materials
  - 1.2.1.1 Towel
  - 1.2.1.2 Bath mat
  - 1.2.1.3 Wash cloth
  - 1.2.1.4 Soap
  - 1.2.1.5 Shampoo
- 1.2.2 Adjust water temperature
- 1.2.3 Turn on exhaust fan

Meriweather Goss, Originator of the Tight Code Concept

...this format includes page numbers and is followed by a Table of Contents. There's even a "WORD PROCESSOR" output format for printing to disk. This format is designed for sending text to WordStar or any compatible word processor.

ERROR-HANDLING

The manual devotes six pages to the program's 29 error messages. I couldn't make the program crash. ThinkTank politely prints "ThinkTank program disk must be in drive A:" if you try to boot with an unauthorized copy of the program disk (hard disk users take note: the ThinkTank program disk HAS to be in drive A: to get underway).

DOCUMENTATION

The 53-page User's Manual is nicely laid out, in two colors with numerous graphics. An index and table of contents are included. Most importantly, a separate Quick Reference Guide is included in the package, clearly defining all of the cursor keys, the function keys, and the Alt-combination keys available in the program. The only fault with the manual lies in its lack of necessity: the program is so obvious that there's rarely any need for the manual.

SUPPORT

Living Videotext, Inc. provides their support line phone number right up front, in ten-point text. It's a toll call. The warranty is for three months; program disks that fail within twelve months after the warranty expires are replaced for $25. The program disk is copy protected: you receive only one with the package; a backup is sent after you return the registration card.

OPINION

ThinkTank does its job extremely well: quickly, completely, and in a manner that requires only a few minutes to learn. Few programs are this well presented, in packaging, documentation, or user interface. ThinkTank is a professional product in every way.

The only rub comes during an examination of ThinkTank's necessity. How often do you require a detailed outline? How well could your word processor do the same thing? There are many people — lecturers, teachers, politicians — who use outlines frequently. For them, ThinkTank's advantages over a word processor may be quite worth it. Others should examine their needs carefully before buying.

*THINKTANK*
*Living Videotext Inc.*
*1000 Elwell Court*
*Palo Alto, CA 94303*
*(415) 964-6300* □



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PERFORMANCE SOFTWARE

SoliDATA is a performance evaluation software program from FASTORE. It allows the user to analyse DOS function calls, monitor file utilization and quantify the system's overall efficiency. It can also compare various storage technologies including floppy and hard disks, bubble memory and the SoliDISK mass storage device. The access time for each media is displayed.

SoliDATA software accumulates data over a period of time recording PC usage patterns by intercepting DOS functions, recording the information, and upon request, generating a detailed hard-copy analysis.

For a limited time, SoliDATA is available for $20. Satisfaction or your money back. Send check or money order to:

*FASTORE*
*200 Danbury Road*
*Wilton, CT 06897*
*(203) 834-1975*

1/2 INCH TAPE TO 5-1/4 DISKETTES AND BACK

CMS offers the big leap: mainframe tape to 5-1/4 diskettes. You send them your half-inch, 9-track, 1600-bpi magnetic tape and they send you the information back on floppy disks. Now you can access all of your name/address files, inventory files, etc., that your large IBM mainframe has been hogging. (Now you know you could do this sort of transfer with a modem setup on your PC and mainframe.) CMS will also convert the files into Mail Merge, dBase, or TIM files. CMSwill also go from 5-1/4 diskettes to half-inch tape. They can convert FORTRAN programs from tape to diskette and guarantee that your operating system will read them. They have a fee structure that is based on the number of bytes converted (which reminds me of an IBM system patch manual).

*CMS*
*P.O. Box 1152*
*Shawnee Mission, KS 66222*
*(913) 384-2006*

SMARTCOM II UPGRADE

Hayes has released an upgraded version of their Smartcom II telecommunications package. They have added quite a few features: terminal emulation, batch file unattended operation, and voice/data switching. They have even (can you believe it) condescended to provide XMODEM transfer protocol so you don't have to switch to PC-TALK every time you want to download an EXE or COM file from someone's BBS. Why in the world all decent communications packages don't include XMODEM protocol I'll never know.

The upgrade is only $25. Call Hayes for your RA (Return Authorization) number and send them your original diskette only.

*Hayes Microcomputer Products*
*5923 Peachtree Industrial Blvd.*
*Norcross, GA 30092*
*(404) 449-8791*

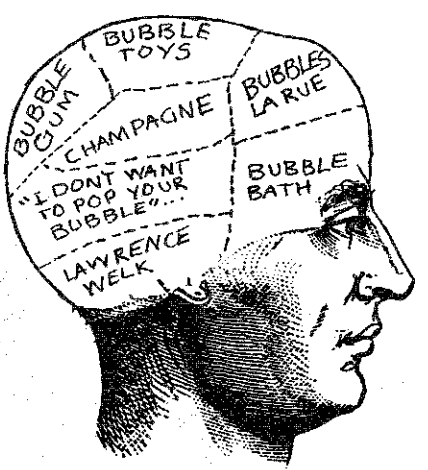
68000 CROSS ASSEMBLER

Avocet has released a cross assembler (XMAC68K) which accepts 68000 code and produces hex object code in a number of formats (Intel, Motorola, and Mostek). The assembler is compatible with Motorola's assembly language, has macro and structured statements, detailed cross referencing, and plain-English diagnostic messages. XMAC68K is priced at $595 and runs under MS-DOS, PC-DOS, CP/M, CP/M-86. Versions for several UNIX systems are scheduled for release in the fourth quarter of 1984.

Avocet has been supplying low-cost microprocessor development tools since 1979. They offer a full range of cross-assemblers for an extensive number of machines.

SCROLL & RECALL

Two features that you find yourself wanting are the ability to scroll backward through text that has already bypassed the screen, and edit a DOS command line that you have entered




BUBBLE MEMORY





PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

incorrectly. Both of these enhancements are part of SCROLL & RECALL offered by Opt-Tech Data. The RECALL function stores your DOS lines as you enter them and can bring any or all of them back to you for editing.

*SCROLL & RECALL*
*Opt-Tech Data Processing*
*P.O. Box 2167*
*Humble, TX 77347*
*(713) 454-7428*

GRAPHICS SCREEN TO SLIDE PRESENTATION

The folks at Polaroid have created the fastest computer display to 35mm slide transfer procedure. You use your 35mm camera and their picture-taking box. Then you develop the film in another black box and insert the finished film in a slide holder.

Perhaps the nicest feature is the ability to select/enhance your graphic screen's color images. The Polaroid system lets you pick and choose brighter and higher resolution colors to give your finished slides a quality look. For more information contact:

*Polaroid Corp.*
*Industrial Marketing Department*
*575 Technology Sq.*
*Cambridge, MA 02139*
*800-225-1618*

## ADVERTISERS


ABComputing ..... 20
Baen Books ..... 13
C Source ..... 11
Cybernetic Micro Systems ..... 31
Data Base Decisions ..... 9-19
DWB Associates ..... 30
Essential Software ..... 25
Greenleaf Software ..... 32
Morgan Computing ..... 28
O.E.S Systems ..... 28
Opt-Tech ..... 10-38
Phoenix Computer Products ..... Cover
Programmers Shop ..... 4
Qualitas ..... 41
Rational Systems ..... 7
Rollin Soft ..... 24
Smartware ..... 42


LOTUS SUPPORT PRODUCT CATALOG

4-5-6 WORLD will send you a free catalog of Lotus 1-2-3 and Symphony support and secondary products. It covers integrated enhancement software, template sets, books, training aids, video tapes, tutorials, hardware and accessories. Numerous applications are presented: accounting, budget, record keeping, sales tools, and asset management. If you have a series of Lotus tools you should drop them a line.

*Lotus 1-2-3 Catalog (free)*
*4-5-6 WORLD*
*Dept. P-101*
*P.O. Box 22657*
*Santa, Barbara, CA 93121*
*(800) 524-5678*

MATRIX PROGRAMMING LANGUAGE

Large-scale number crunching can now be done on the IBM PC with GAUSS, a sophisticated matrix programming package. Skip the mainframe bottleneck and do it yourself on a PC. It is an excellent programming environment for statistics, econometrics, engineering, simulations, linear and non-linear equations, differential equations and curve fitting. The 8087 is used for computation, and high-precision floating point arithmetic is provided to 19 digit (80 bit) accuracy. (Standard double precision is only 64 bits). GAUSS uses numerical algorithms designed to minimize rounding error. It is an environment, i.e., it includes a built-in editor with a command mode for interactive programming.

*GAUSS*
*Applied Technical Systems*
*P.O. Box 6487*
*Kent, WA 98064*
*(206) 631-6679*

ULTIMATE OPTIMIZING TOOL

A full-feature program optimizer, PROFILER, has been written by DWB and Associates of Beaverton, Oregon. It allows the programmer to examine where his compiled programs spend most of their time. The general rule is that 85% to 95% of program time is spent executing 10% of the code. If you can isolate which 10% is being repeatedly being executed, you can optimize it for speed. PROFILER samples the instruction pointer at a fixed interval. You can divideor partition your memory in up to 16 different areas. The areas can be of any size. PROFILER produces a report to tell you how much time your program spends in each of these areas. This is a program development tool for software writers.



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

CODE CORRECTIONS

Corrected code from the article C Utility Routines, Vol. 2 #3, by Warren Hillman. For more information on the extensive set of C Building Block routines, contact:

*Novum Organum*
*29 Egerton Road*
*Arlington, MA 02174*
*(617) 641-1650*

---

```
/**********************
© 1984 Novum Organum
/* #include 'basic.rpl', not needed */
#define EOS '+0' /* End Of String, NULL */
/* The following routine separates the pathname or disk
specifier from its filename, and returns it in a separate array.
We do this by initially copying the combined name into
another array, then searching for a colon or slash that ends
the pathname. So one array ends up holding the pathname
with a NULL stuffed in there to end the string, and the other
array has its starting point (pointer) moved to show where
the filename begins.

Only a pointer to the filename array is returned since you
already know the address of the pathname array. */

char *StripPath (out,in)
    char    *out;
    char    *in;
{
    char    *fil_nam, *endpath;

    strcpy (out,in);        /* duplicate, for working, */
    fil_nam = in;           /* pointer to start of name, */
    endpath = out;          /* and end-of-path pointer to stuff NULL */

    while
        (*in)
        {
            /* look for slashes or colons, */
        if    (*in == ':')
            {                       /* if a colon, */
            endpath= ++out;         /* point to name, */
            fil_nam = ++in;
            }
        else
        if    (*in == '/')
            {                         /* if we find a slash, */
            endpath= ++out;            /* move the pointer up, */
            fil_nam = ++in;
            }
        else
        if    (*in == '\\')
            {
            endpath= ++out;         /* keep moving it up, */
            fil_nam = ++in;
            }
        else
            {
            ++in;
            ++out;
            }
        }

    *endpath = EOS;           /* delete the name part, */

    return(fil_nam);      /* return ptr to name part. */
}
```

□



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

# ARE YOU AN ISSUE SHY OF A FULL SET?

Complete your library! Order any back issues you may have missed — or become a new subscriber. Use the form below or the handy subscription envelope provided. You'll be glad you did.

- Vol. 1, No. 1: Norton Pascal I/O, Program Style Manual, COBOL, System Analysis, BASIC modules, Rounding error.
- Vol. 1, No. 2: Defining character sets, C86 BIOS calls, Copy protection, Pascal code, Input routines, Writing an Assembler.
- Vol. 1, No. 3: ASM dumb terminal, C source code, ASM graphics line drawing, Text editors reviewed, Utilities.
- Vol. 1, No. 4: Diskette directory via Pascal, MASM WHILE-WEND, C/ASM interface, ASM stay resident routine, Structured programming.
- Vol. 1, No. 5: Multi-tasking software review, Transporting C Programs, Programmers calculator, dBase bargraphs, Utilities.
- Vol. 1, No. 6: ASM printer configuration, DOS patch utility, Hi-speed graphics, Software ownership, dWINDOW review.
- Vol. 2, No. 1: Freelance consulting, Sorts in C, Pascal text editor source code, Designing national database, Ultra utilities.
- Vol. 2, No. 2: Understanding C through Assembly language, Utility routines in C and Assembly, Music in C (using a C music program), Cornucopia of software development tools.
- Vol. 2, No. 3: Software development environments, Creating a productive screen handling tool, Generating graphics within windows, Create, edit and play your musical ideas, Linking Assembler to Pascal and Basic.
- Vol. 2, No. 4: Writing a custom ROM BIOS for PC-DOS, The future of disk protection, Time-saving MASM Macros, C routines you can purchase.
- Vol. 2, No. 5: Scroll control using BASIC and Assembler, RTool, a PC-resident programmer's toolkit, Calling Assembly routines from dBase II, Pascal's pointer power.

---

Name _____

Company _____

Address _____

City _____ State _____ Zip _____

Telephone _____

Send me the following back issues:
$4 each U.S., ($5 Canada, $7 Other, U.S. funds) pre-paid

_____ Vol. 1, No. 1
_____ Vol. 1, No. 2
_____ Vol. 1, No. 3
_____ Vol. 1, No. 4
_____ Vol. 1, No. 5
_____ Vol. 1, No. 6
_____ Vol. 2, No. 1
_____ Vol. 2, No. 2
_____ Vol. 2, No. 3
_____ Vol. 2, No. 4
_____ Vol. 2, No. 5

☐ 1 year bi-monthly subscription:
$24.00 ($30 Canada, $40 other, U.S. funds only)

☐ Please renew my subscription

☐ Check enclosed ☐ Bill me (subscriptions only)

☐ Charge my Bankcard: ☐ VISA ☐ MC ☐ Am Ex

# _____ Expir. Date _____

Signature _____

Send to:
PROGRAMMER'S JOURNAL
P.O. Box 30160
Eugene, OR 97403
(503) 345-3043

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PERISCOPE — BUILDING A HARDWARE BOARD FOR THE IBM PC

AT&T SURPRISES ALL

dBASE II TO dBASE III?

INTEL 80286 BUGS BY FNE

CALENDAR FUNCTIONS FOR BASIC

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED